# Dédicaces

*A mes parents,*
*A mes fréres et Sœurs,*
*A toute la famille,*
*A mes amis et collègues,*

JE remercie avant tous Dieu qui m'a donné la force et la volonté pour réaliser ce modeste travail.

Je tiens à exprimer ma sincère gratitude à $M^r$ Francois Charoy, mon directeur de thèse , qui m'a dirigé et conseillé durant toute la période de ma thèse. Sa compétence, son encouragement m'ont été très bénéfiques et m'ont beaucoup aidé dans la réalisation de ce travail.

Je remercie tous nos enseignants du département informatique de l'université de Béjaia.

Ma gratitude va aussi aux membres de jury d'avoir accepté de juger mon travail.

Je remercie toutes les personnes qui m'ont aide de prés ou de loin. Merci à $M^{elle}$ IMLOUL Salima pour ses conseils.

Ma reconnaissance va bien sûr à ma famille en général, et en particulier mon père qui m'a aidé et a été à mes cotés. Je te remercie Thada.

Sans oublier tous mes amis(es) et particulièrement mes collègues de l'école doctorale de Bejaia.

**Merci à tous.**

# Liste des Acronyms

**BP** **B**usiness **P**rocess
**SGWF** **S**ystème de **G**estion de **W**ork**f**low
**Wf** **W**ork**f**low
**WfMS** **W**ork**f**low **M**anagement **S**ystem
**LTL** **L**inear **T**emporal **L**ogic
**CSP** **C**onstraint **S**atisfaction **P**roblem
**TCN** **T**emporal **C**onstraint **N**etworks
**PA** **P**oint **A**lgebra
**BPCN** **B**usiness **P**rocess **C**onstraint **N**etworks
**RDP** **R**éseaux **D**e **P**étri

# Table des matières

# Liste des figures

# Liste des tableaux

# Introduction générale

L'AUTOMATISATION des fonctions spécifiques d'une entreprise est devenu un des axes de recherche très important dans le domaine de l'organisation des activités et des procédures du travail. Il nécessite un processus de modélisation et d'automatisation rigoureux qui permet de contrôler et d'évoluer l'ensemble de tâches participantes à la définition du processus considéré ou ce qu'on appelle un **processus métier( Business Process (BP))** .
La technologie workflow est délivrée comme une solution à ce type de processus.

La flexibilité offerte dans la spécification de workflow permet une modélisation partielle au préalable et ainsi une certaine liberté vis à vis de l'utilisateur dans sa spécification flexible (dans le choix de ses activités et contraintes), et vis à vis du système dans l'exploitation de l'univers des solutions possibles.

Néanmoins, cette flexibilité sans un certain degré de contrôle cause de véritables problèmes dans l'exécution, en ce qui concerne les erreurs qui peuvent générer des instances incorrectes ne satisfaisant pas le but du processus métier considéré.

Notre objectif dans ce travail est d'éviter les erreurs causées par une modélisation flexible en combinant les deux principes (la flexibilité et le contrôle) dans un seul formalisme par l'ajout de ce qu'on a appelé **points de contrôle** et la création de **poches impératives**. Cela permet d'avoir un modèle dont il existe des parties de processus qui doivent être contrôlées et d'autres qui doivent être flexibles.

Pour atteindre notre objectif, il est primordial d'étudier les deux approches de modélisation de workflow qu'on trouve dans la littérature.

1. **L'approche impérative** : C'est une approche strictement contrôlable et qui permet une définition du modèle au préalable comme dans le cas des réseaux de pétri. Elle ne permet aucun changement à l'exécution et de ce fait aucune possibilité de changer un modèle. Cela rend presque impossible à l'utilisateur

de spécifier librement un modèle et ainsi diminue les possibilités de flexibilité [18] [**?**] [27].

2. **L'approche déclarative** : Qui consiste à définir le modèle comme étant une formule de logique temporelle (LTL) qui est à son tour une conjonction de sous-formules LTL. Cette formule permet de décrire un processus en utilisant ses caractéristiques et c'est à l'ordinateur d'explorer l'univers des solutions vraies possibles [28]. La flexibilité de cette approche consiste en la liberté qu'elle offre dans la spécification et aussi le nombre d'alternatives. Néanmoins, elle permet moins de contrôle en ce qui concerne la correction des instances produites.

## • Objectifs et Motivations

Le but du travail est d'arriver à avoir un compromis entre la flexibilité des modèles déclaratifs et le contrôle des modèles impératifs.

A partir d'un modèle entièrement flexible décrit avec un ensemble de contraintes représentées avec une formule LTL, on essaye de capter un comportement impératif (séquences séries ou parallèles) afin de le cerner et de le contrôler par des parties impératives en utilisant la force de l'algorithme de chemin consistant et les relations d'intervalle d'Allen [27] [9], et de ce fait avoir notre modèle hybride.

La force de l'approche est basée sur la définition d'un ensemble de tâches constituant le processus métier (ou la partie du processus à modéliser), et une formule logique LTL. En ce basant sur la définition des templates donnés dans DecSerFlow (voir ConDec et Declare [31] ), une formule LTL sera représentée dans le modèle déclaratif de départ avec une syntaxe bien définit utilisée dans les modèles déclaratifs tel que DecSerFlow [40] , ConDec ,Declare [33] [31], *(la formule LTL $G(x \Rightarrow y)$ est équivalente à la formule de relation dans DecSerFlow qui est : Precedence(x,y) qui signifie à chaque exécution de y, x doit exécutée avant)* . Cette définition syntaxique permet d'aider l'utilisateur ne connaissant pas le langage LTL à spécifier ces contraintes plus aisément.

Avec cette notation on peut définir notre modèle déclaratif de départ D comme suit : D=$(T, \theta)$. Avec :

T= $\underset{i}{t}$, i=1..N. Avec N est le nombre de tâches du processus.
$\theta = \underset{j}{\wedge}\theta$ avec $\underset{j}{\theta}$=contrainte . j =1..M. Avec M est le nombre de contraintes du processus.

- **Structure du mémoire**

Ce mémoire est organisé comme suit :

Le premier chapitre est une introduction à quelques concepts liés au domaine de processus métier et workflow, et cela afin d'encapsuler et de comprendre la signification de chaque concept dans la technologie workflow et spécification de processus métier.

Le deuxième chapitre est un état de l'art sur les deux méthodes de modélisation de workflow, à savoir la méthode déclarative par la logique temporelle linéaire LTL et la méthode impérative par réseaux de pétri. Dans ce chapitre nous discuterons d'une part sur la flexibilité offerte par les modèles déclaratifs et d'une autre part sur le contrôle offert par les modèles impératifs.

Dans le troisième chapitre, on étudie la flexibilité pour les workflows, l'apport de la flexibilité dans la spécification de workflow et son incorporation par l'ajout de concept des poches de flexibilité.

Le quatrième chapitre présente notre approche de modélisation hybride de workflow dont la problématique est le manque de contrôle dans le modèle déclaratif ce qui entraine des erreurs d'exécution. L'approche est constituée de plusieurs étapes qui effectuent des transformations sur un modèle déclaratif de départ .

Dans le cinquième chapitre on a décrit l'implémentation et la mise en œuvre d'un exemple de processus métier proposé par Pesic et Aalst, ainsi qu'un exemple comparatif d'une modélisation du même exemple du processus par un réseau de pétri [33].

Notre mémoire s'achève par une conclusion générale résumant les grands points qui ont été abordé ainsi que des perspectives que nous souhaitons accomplir prochainement.

# Chapitre 1

# Terminologie de base

## 1.1 Introduction

DANS le monde de l'entreprise, l'organisation du travail est un problème complexe. Le concept du processus métier a recueilli un intérêt croissant dans le domaine d'entreprises dans le but de comprendre mieux ce qu'elles font et comment elles peuvent le faire plus efficacement.

Des processus métiers appelés souvent Business Process (BP) sont de plus en plus orientés en tant que travail coopératif que les compagnies ou les entreprises doivent contrôler et maintenir si elles doivent continuer à fonctionner efficacement.

Dans le présent chapitre, nous présentons quelques concepts de base liés aux technologies d'automatisation de processus d'entreprise en générale, et les processus métiers en particuliers et spécification de ces derniers.

## 1.2 Un processus métier ou Business Process

Il existe plusieurs définitions de processus métier dans la littérature [34] [23] [10] [41],Voici quelques une :

Hammer et Champy [23] ont définit le processus métier comme étant un ensemble d'activités qui s'exécutent ensemble pour fournir un produit de valeur au client.

Selon Denis Berthier [10], un processus métier est définit comme un ensemble d'activités pouvant être considérées comme une succession de transformations (dont le résultat est de répondre aux besoins du client), son déroulement peut être indiqué par des transitions éventuellement caractérisées par des événements. L'agencement des activités correspond à la structure du processus.

Toutes ces définitions focalisent sur le fait qu'un processus métier est un ensemble de tâches qui s'exécutent selon un ordre quelconque dans le but de produire un résultat et atteindre un objectif commun. Il correspond à l'automatisation d'un processus métier et considéré comme synonyme du mot *workflow* ou flux de travail [42] qui est au centre de notre intérêt dans ce présent mémoire.

## 1.3 Modélisation des processus métiers

La notion de modèle recouvre toute représentation d'un système réel, exprimée dans un langage spécifique, le plus souvent sous forme graphique. C'est une abstraction du processus qui sert comme base pour une étude détaillée du processus en considération [14]. Elle doit être compréhensible d'une façon claire et transparente.

Pour un processus métier c'est la description des activités d'une organisation en termes de tâches, agents, rôles et procédures. C'est un ensemble de techniques utilisées pour représenter et structurer la connaissance du métier d'une entreprise. La principale difficulté consiste à restituer une représentation suffisamment pertinente et significative de la réalité pour obtenir le résultat voulu.

## 1.4 Workflow ou l'automatisation de processus métier

C'est récemment que le monde de l'entreprise s'est intéressé au rôle que peut jouer l'informatique dans l'organisation du travail. La technologie workflow a été considérée comme une technologie qui automatise les processus métiers pour fournir une vue et une compréhension globale des modèles de processus métier.

### 1.4.1 Définition de Workflow

Plusieurs définitions existent pour ce concept de workflow [6] [20] [21] [39] [35] :

#### 1.4.1.1 Définition 1

On appelle un **Workflow (Wf)** l'automatisation complète ou partielle des procédés durant lesquels des informations sont passées d'un participant à un autre et des tâches sont effectuées par ces derniers en accord avec des procédures.
*Assurer que le bon travail est fait au bon moment par la bonne personne et dans le bon ordre* [6].

#### 1.4.1.2 Définition 2

Un workflow est défini comme une suite ordonnée d'activités dont le but est la résolution d'un problème ou la réalisation d'un objectif [21]. Chaque activité représente une étape logique dans la résolution. Elles sont ordonnées par des transitions [16].

#### 1.4.1.3 Définition 3

Dans cette définition, le terme *WorkFlow* (traduit littéralement *flux de travail*) est la modélisation et la gestion informatique de l'ensemble des tâches à accomplir par les différents acteurs impliqué dans la réalisation d'un processus métier (aussi appelé processus opérationnel). Le terme de Workflow pourrait donc être traduit en français par ***Gestion électronique des processus métier*** [20].

Un workflow peut donc être défini comme un modèle informatique pour représenter un processus métier, il implique un nombre limité d'acteurs devant accomplir en un temps limité, des tâches afin de satisfaire un objectif global.

### 1.4.2 Représentation de workflow

Un workflow est considéré comme un graphe qui comporte des nœuds et des arcs. Une relation de contrôle de flux permet de relier deux nœuds dans le graphe et visualise leur ordre d'exécution. Les noeuds sont les constructions de bases d'un Workflow représentant les tâches ou les coordinateurs qui sont : choix *(choice)*, fusion *(merge)*, branchement *(fork)* et synchronisateur *(Synchronizer)*, et les deux nœuds de début *(begin)* et de fin *(end)*.
Voici un exemple d'un graphe workflow.

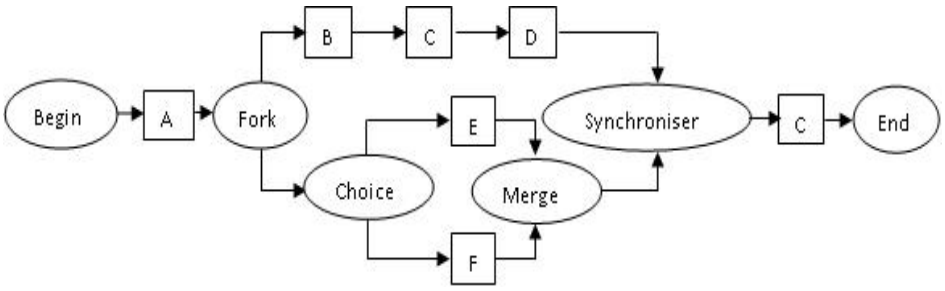



FIG. 1.1: Exemple d'un graphe de Workflow [29].

### 1.4.3 Definition formelle de workflow

Selon Maria Orlowska [28], Sadiq et al [36], un workflow est définit comme un graphe orienté $W =< N, F >$ . Avec N ensemble fini de nœuds et F ensemble fini de relations de flux f= N x N.

Un nœud n ($n \in N$) peut être une tâche t ($t \in T$) représentée avec un rectangle ou un coordinateur c ($c \in C$) représenté avec un cercle ou un losange.

$\forall n \in N$, TypeNœud : $n \rightarrow \left\{ \text{ Coordinateur, Tâche } \right\}$.
- C = ensemble fini de coordinateurs.
- T = ensemble fini de tâches (activités).
- N = $(C \cup T) avec (C \cap T) = \emptyset$.
- $\forall (c \in C)$ TypCoordinateur : $\rightarrow \left\{ \begin{array}{c} AND - Split, AND - Join, \\ XOR - Split, XOR - Join, Begin, End \end{array} \right\}$ [28]

La figure suivante est une représentation graphique d'un processus workflow avec les différents nœuds coordinateurs et tâches.

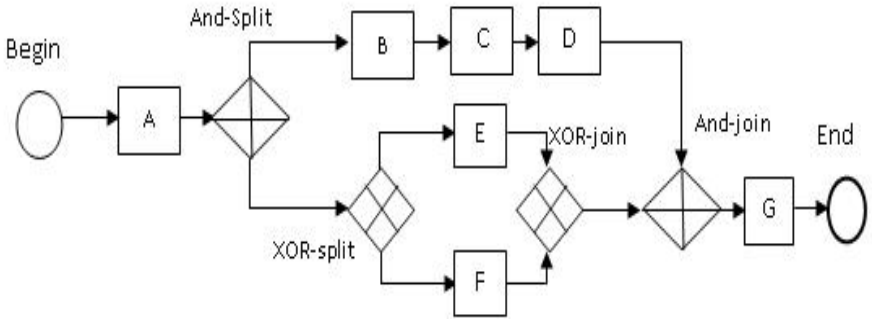



FIG. 1.2: Constructions de base d'un Workflow [28].

### 1.4.4 Une représentation correcte d'un workflow

La vérification de la correction d'un schéma de workflow consiste à vérifier l'absence de conflit structurel. Selon Sivaraman et al [14], ont définit des critères de correction d'un workflow comme :

- **Critère 1** ***Le non blocage de workflow*** : Un workflow est sans conflit structurel de blocage s'il ne génère pas une instance qui contient un sous ensemble de nœuds entrant perdus à un nœud and-join (synchroniser). Le schéma suivant illustre ce comportement.

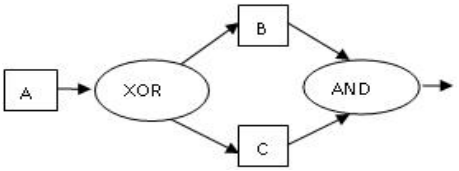



FIG. 1.3: Exemple d'un conflit structurel de blocage dans un Workflow [14].

- **Critère 2** ***Le non manque de synchronisation*** : Un workflow est sans conflit structurel de manque de synchronisation s'il ne génère pas d'instances qui contient plus qu'un nœud entrant à un nœud xor-join (merge). Le schéma suivant illustre ce comportement.

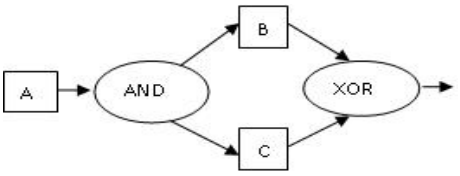



FIG. 1.4: Exemple d'un conflit structurel de manque de synchronisation dans un Workflow [14].

### 1.4.5 Type de WorkFlow

La technologie de workflow est appliquée dans des secteurs tout à fait divers. Cette diversité a eu un impact fort sur la recherche de Workflow . On peut trouver dans la littérature plusieurs type de Workflow [21] [20].

#### 1.4.5.1 Le Workflow de production

Il correspond à la gestion des processus de base d'une entreprise. Les procédures supportent peu de changements dans le temps et les transactions sont répétitives. On peut y trouver par exemple la production de contrats d'assurance, la gestion de réclamations des clients .... etc. [20].

#### 1.4.5.2 Le Workflow administratif

Il correspond à tout ce qui est routage de formulaires, basé en général sur une infrastructure de messagerie.

#### 1.4.5.3 Le Workflow ad-hoc

Ce type de workflow est utilisé pour la gestion des procédures non déterminées, ou changeantes.

#### 1.4.5.4 Le Workflow coopératif

Il gère des procédures évoluant assez fréquemment et liées à un groupe de travail restreint dans l'entreprise.

## 1.5 Moteur de workflow

Se charge d'exécuter le modèle tout en tenant compte des contraintes de la plate-forme sous laquelle il s'exécute. Pour exécuter un modèle le moteur crée une instance de modèle de Workflow.

Le moteur de Workflow peut gérer l'exécution simultanée de plusieurs instances du même modèle de Workflow, ce qui permet le passage à l'échelle ainsi qu'une bonne réutilisation des modèles de Workflow.

## 1.6 Système de gestion de Workflow (SGWF)

**W**ork**f**low **M**anagement **S**ystem (WfMS). Il définit et gère l'exécution d'un ou de plusieurs workflow à l'aide d'un environnement logiciel fonctionnant avec un ou plusieurs moteurs de workflow et capable d'interpréter la définition d'un processus, de gérer la coordination des participants et d'appeler des applications externes. Un SGWF implique deux phases [14] :

- **La phase de modélisation** : Donne une abstraction des procédures métiers et définit les caractéristiques implémentables pour le workflow.
- **La phase d'exécution** : Exécute les instances de workflow.

La figure suivante représente les deux phases du SGWF.

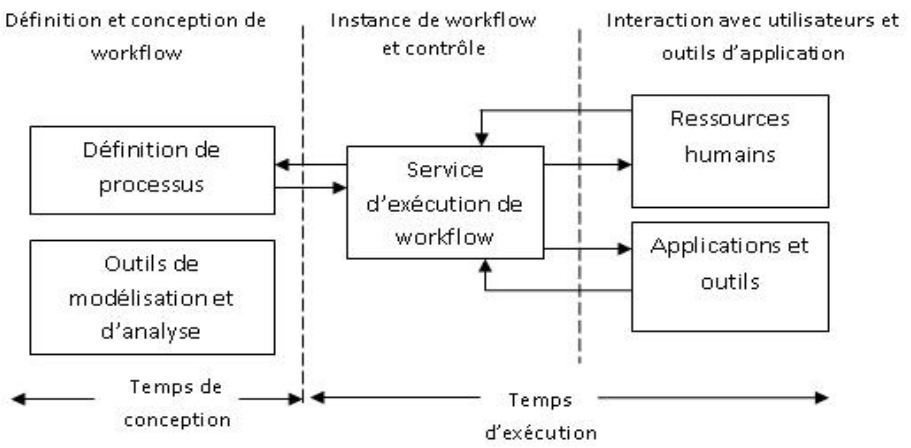



FIG. 1.5: Système de Gestion de Workflow Modèle de référence [14].

## 1.7 Standards de workflow

Pour permettre l'intégration de standards dans le monde des systèmes de workflows, des éditeurs de logiciels, des laboratoires de recherches et des utilisateurs de ces systèmes ont créés le consortium Workflow Management Coalition (WfMC). L'objectif de cette association est la promotion et le développement des systèmes de workflows. Pour cela, ils ont défini un modèle de référence (figure 1.6.) centré autour du moteur d'exécution. Ce modèle présente 5 interfaces de standardisations [16].

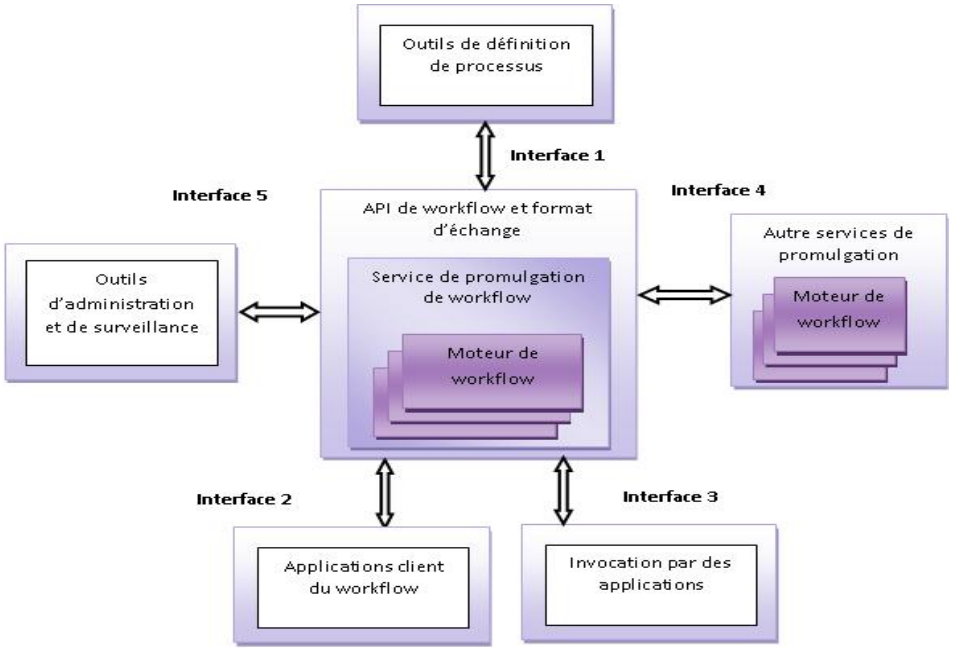



FIG. 1.6: Modèle de référence des systèmes de workflow [16]

- **Interface 1** : Elle correspond à l'échange des modèles entre moteurs de workflow et les différents outils de modélisation de processus. Cette interface est désignée sous le terme d'interface import-export de définition de processus qui devrait fournir un format d'échange commun à plusieurs types d'informations.
- **Interface 2** : Elle permet à des applications clientes de communiquer avec le moteur de workflow.
- **Interface 3** : Elle permet au système de workflows d'appeler des applications clientes. Les systèmes de gestion de workflow doivent communiquer avec toutes les applications externes nécessaires à l'accomplissement des tâches.
- **Interface 4** : Elle permet l'interopérabilité entre moteurs de workflows.
- **Interface 5** : Elle correspond à l'interaction entre les applications d'administration et le moteur de workflow. Il s'agit de définir un standard d'interface permettant à un outil d'administration et de pilotage de travailler avec n'importe quel moteur de workflow.

## 1.8 Conclusion

Un workflow est une solution informatique qui n'est rien de plus qu'un support à la réorganisation d'un processus métier, mais qui est très importante et qui reste un moyen incontournable si on veut augmenter le rendement des entreprises et satisfaire les exigences du client.

Dans ce chapitre, nous avons abordé quelques concepts de base liés à la technologie de workflow et modélisation de processus métier. L'apport de définition de workflow pour l'automatisation et l'amélioration des processus d'entreprise dans le but de faire face aux changements du marché et satisfaction des exigences des clients.

Afin de comprendre mieux le fonctionnement des processus métiers et de les automatiser d'une manière plus efficace, des méthodes de modélisation ont été proposées dans la littérature. Dans le prochain chapitre, nous présentons une étude détaillée sur les deux approches de modélisation de workflow à savoir l'approche déclarative et l'approche impérative.

# Chapitre 2

# Etat de l'art sur les deux méthodes de modélisation de Workflow

## 2.1 Introduction

MODÉLISER les workflow est la tâche de créer des spécifications qui permettent la représentation du fonctionnement et du comportement du système (partie). De telles spécifications seront habituellement utilisées comme des entrées dans un **S**ystème de **G**estion de **W**ork**f**low (SGWF).

L'utilisation des spécifications formelles de WF a été souvent préconisée ; les méthodes formelles réduisent l'ambiguïté et ouvrent des possibilités pour la vérification et l'analyse.

Différents formalismes ont été suggérés pour modéliser les WF, les deux modélisations existantes dans la littérature sont le sujet du présent chapitre. Une première partie sera consacrée à la spécification impérative ou la modélisation par les **R**éseaux **D**e **P**étri (RDP) qui permet une spécification contrôlable au préalable, la deuxième sera consacrée à la spécification déclarative et plus précisément l'utilisation de la logique temporelle linéaire (**L**inear **T**emporal **L**ogic (LTL)) qui permet une spécification beaucoup plus flexible.

## 2.2 Approche impérative

### 2.2.1 Introduction à l'approche impérative

La modélisation impérative de workflow est la tâche de spécification de processus métiers à l'aide d'un langage formelle de modélisation dit impératif (procédural, opérationnel).

Les langages impératifs permettent une spécification a priori du processus modélisé en indiquant explicitement le procédé *comment*, C'est à dire spécifier clairement comment atteindre le but du processus en considération. Un exemple de ce type de modélisation est donné dans la figure suivante :

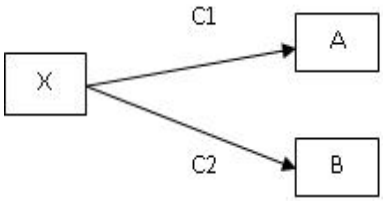



FIG. 2.1: Exemple d'une modélisation impérative.

Dans cette figure, une activité de décision X est introduite. Elle doit être exécutée à un instant particulier et exige des conditions C1 et C2 pour le choix de l'une des activités A ou B. Dans cet exemple on voit bien que le comportement du système est bien spécifié à l'avance et que l'exécution de A ou de B est conditionné par la satisfaction de l'une des conditions C1 ou C2.

Dans ce qui suit nous discutons d'une méthode impérative de modélisation basée sur le formalisme de réseaux de pétri ainsi que son utilisation pour la modélisation des systèmes dans le contexte de processus métiers et de la gestion de workflow [3] [32] [7].

### 2.2.2 Les réseaux de pétri

Les réseaux de pétri ont été introduit par Dr.Karl Adam Petri en 1962. Ce sont des formalismes graphiques et mathématiques pour la modélisation et l'analyse des systèmes dynamiques. Ils sont appliqués dans plusieurs secteurs comme les protocoles de communication, l'évaluation des performances, les systèmes distribués ...etc.

#### 2.2.2.1 Réseaux de pétri classiques

Un réseau de pétri classique est un graphe biparti orienté (bipartite directed graphs) qui possède deux types de nœuds ; les places et les transitions. Les arcs orientés connectent les transitions aux places ou les places aux transitions. Afin d'étudier le comportement et le changement d'états d'un système modélisé avec les RDP, chaque place peut contenir potentiellement zéro ou plusieurs jetons [24].

Il existe dans la littérature plusieurs définitions des réseaux de pétri [3] [32] [24] [17] [11]. Voici l'une des définitions formelle d'un Réseau de pétri classique :

**Définition :**
Un réseau de pétri N est un triplet (P, T, F) avec :

- P : est un ensemble fini de places.
- T : est un ensemble fini de transitions.
- F $\subseteq$ (T x P) $\cup$ (P x T) est un ensemble d'arcs ou de relations de flux [3] [4].

- Une place **p** est dite place d'entrée (input place) de la transition **t** Ssi, il existe un arc orienté de **p** vers **t**.
- Une place **p** est dite place de sortie (output place) de la transition **t** Ssi, il existe un arc orienté de **t** vers **p**
- Une transition **t** est dite transition d'entrée de la place **p** Ssi, il existe un arc orienté de **t** vers **p**
- Une transition **t** est dite transition de sortie de la place **p** Ssi, il existe un arc orienté de **p** vers **t**
- On appelle prédécesseur d'une transition **t (*t)** l'ensemble des places d'entrées de **t** : ***t** = $\left\{p \in P/ \text{ p est une place d'entrée de } \mathbf{t}\right\}$.
- On appelle successeur d'une transition **t (t*)** l'ensemble des places de sorties de **t** : **t***= $\left\{p \in P \text{ / p est une place de sortie de } \mathbf{t}\right\}$.
- On appelle prédécesseurs d'une place **p (*p)** l'ensemble des transition d'entrées de **p**. ***p**= $\left\{t \in \mathbf{T} \text{ / t est une transition d'entrée de } \mathbf{p}\right\}$.
- On appelle successeurs d'une place **p ( p*)** l'ensemble des transition de sorties de **p**. **p*** = $\left\{t \in \mathbf{T} \text{ / t est une transition de sortie de } \mathbf{p}\right\}$.
- Une transition **t** est permise *(enabled)* Ssi dans toutes ses places d'entrée il existe au moins un jeton.
- Une transition t peut être activée *(fire)* si elle est permise. Et quand elle est activée, elle consomme un jeton dans chaque place d'entrée et produit un jeton dans chaque place de sortie.
- A n'importe quel moment une place peut contenir zéro ou plusieurs jetons. La distribution de ces jetons à travers les places représente un état du système.

C'est ce qu'on appelle un marquage (on le note souvent pas M) M : P $\rightarrow \mathcal{N}$ (le nombre de jetons dans chaque place du réseau de pétri).

Voici un exemple simple d'un réseau de pétri classique :

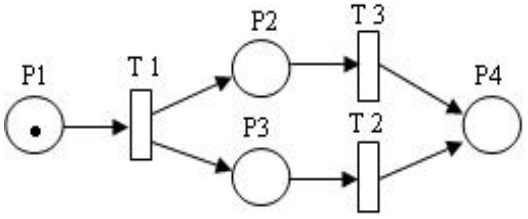



FIG. 2.2: Un simple réseau de pétri [24].

Dans cet exemple le jeton de la place P1 rend la transition T1 permise et peut être activée, son activation produit deux jetons dans ses deux places de sorties (P2, P3).

#### 2.2.2.2 Les réseaux de pétri à haut niveau

Les réseaux de pétri permettent une modélisation des états du système, événements, conditions, synchronisation, itération, parallélisme, choix. Cependant la modélisation des systèmes réels avec les RDP classique est une tâche difficile et complexe à cause manque de notions de temps et de données [3] .

Plusieurs extensions ont été développées fournissant de nouvelles constructions par rapport aux réseaux de pétri classiques mais en gardant la même expressivité, on peut trouver :

- Une extension avec la couleur pour modéliser les données (réseaux de pétri colorés) [32] [17] .
- Une extension avec le temps (réseaux de pétri temporels) [38].
- Une extension avec une hiérarchie pour la structuration de grands modèles.

### 2.2.3 Modélisation de workflow avec les réseaux de pétri

Plusieurs recherches ont recommandés l'utilisation des réseaux de pétri pour la modélisation de workflow, et cela à cause de plusieurs facteurs [32] [8].

- **Une sémantique formelle** : Rend la spécification de workflow non ambigüe. L'interprétation de ces spécifications peut être définie mathématiquement.

- **Techniques d'analyse** : Ils existent plusieurs techniques d'analyse de propriétés qualitatives et quantitatives de modèle de workflow à base des réseaux de pétri, une des propriétés qualitatives est de vérifier l'occurrence d'un inter blocage en vérifiant l'accessibilité de toutes les tâches du réseau [32] .
- **Expressivité** : Les réseaux de pétri peuvent exprimer la structure des processus ainsi que leurs dynamisme, y compris la concurrence (le parallélisme). Ils servent aussi comme langage pour la définition de la structure d'un processus, simulation de processus et la gestion de workflow [22].

#### 2.2.3.1 Workflow à base de réseaux de pétri (WF-net)

La modélisation de workflow en terme de réseaux de pétri consiste à modéliser le comportement dynamique du système en représentant les tâches sous forme de transitions, les conditions sous forme de places et les cas sont représentés par les jetons. Les réseaux de pétri qui permettent de modéliser les workflows sont appelés WF-net (réseaux de workflow).

**Définition (WF-nets)** : Le réseau de pétri qui permet de modéliser le contrôle de flux d'un workflow est appelé un WF-net [26] [2] [25] (réseaux workflow) Ssi :

1. Il existe une seule place source $\mathbf{i} \in \mathbf{P}$ avec $^*\mathbf{i} = \emptyset$ . (**i** début du processus).
2. Il existe une seule place puits **o** $\in$P avec $\mathbf{o}^* = \emptyset$ (**o** est la fin du processus).
3. Chaque nœud x $\in$ (P$\cup$ T) est sur un chemin de **i** à **o**. C'est à dire : Chemin (**i**, x) $\wedge$ Chemin (x, **o**).


Fig. 2.3: Procédure modélisée avec le WF-net [2].

Un réseau de workflow (WF-net) est un réseau de pétri qui possède une seule place d'entrée (i) et une seule place de sortie (o) et puisque n'importe quel cas traité avec la procédure représentée par un WF-net est crée au moment de début de son traitement par le système de gestion de workflow(SGWF) est supprimé une fois que sont traitement est complètement achevé par le SGWF. Cela veut dure que le WF-net spécifie le cycle de vie d'un cas du processus modélisé [5] [4].

**Remarque** : Un chemin **C** dans un WF-net d'un nœud $n_1$ au nœud $n_k$ est une séquence $<n_1, n_2, \ldots, n_k>$ tel que $<n_i, n_{i+1}> \in$ F , pour i=1 ...k-1. F est la relation de flux [38] .

**Propriétés d'un WF-net** : Voici quelques propriétés d'un réseau de workflow.

1. **Solidité (soundness property) :** Le traitement d'un cas commence au moment où un jeton est présenté dans la place i et se termine au moment où on a un jeton dans la place puits o. Cette propriété réfère une instance d'exécution d'un workflow [13].

   *Pour n'importe quel cas, le procédé sera terminé par la suite et le moment où le procédé se termine il y a au moins un jeton dans la place de sortie o et tous les autres places sont vides [1] [13] [25].*

2. **Sain (sound proprety) :** Un WF-net est sain Ssi :
   **i)** Pour chaque état **M** accessible à partir de **i**, il existe une séquence active de transitions menant de **M** vers **o**. Formellement on note :

   $$\forall M(i \rightarrow^* M) \Rightarrow (M \rightarrow^* o)$$

   **ii)** L'état **o** est le seul état accessible de **i** avec au moins un jeton dans la place **o**. Formellement :

   $$\forall M(i \rightarrow^* M \wedge M \geq o) \Rightarrow (M = o)$$

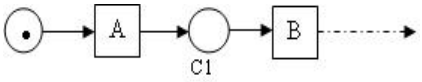
   **iii)** Il n'existe pas de transition morte dans (PN, i). avec PN=Petri Nets (réseaux de pétri). Formellement :

   $$\forall_{\tau \in T},\ \exists_{M,M'}(i \rightarrow^* M \rightarrow^{\tau} M')$$

#### 2.2.3.2 Constructions du routage dans un WF-net

Un réseau de workflow (WF-net) est utilisé pour spécifier le routage de flux dans les cas du workflow, quatre types de routage ont été identifiés :

**Séquentiel :** Utilisé pour traiter la relation de causalité entre les tâches. Soit deux tâches A et B, si la tâche B et exécutée après l'accomplissement de A, ce comportement peut être modélisé avec un RDP en ajoutant les places pour capter les relations de causalité entre les tâches A et B. La figure suivante illustre ce comportement.

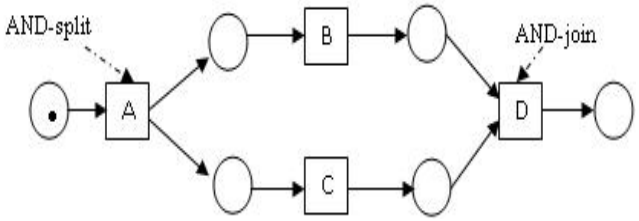



Routage séquentiel.

La place c1 modélise la relation de causalité entre les tâches A et B.

**Parallèle :**

Utiliser dans le cas où l'ordre d'exécution est moins strict. Par exemple si on a deux tâches B et C qui doivent être exécutées mais dans un ordre arbitraire, la modélisation de ce comportement de routage parallèle comporte deux constructions le AND-split et AND-join.

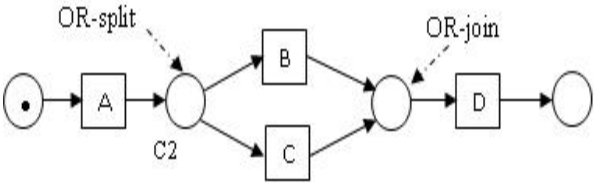



Routage parallèle.

**Conditionnel :**

Utilisé pour traiter les cas où le routage de flux peut dépendre des données d'un cas. Il existe deux types de routage conditionnel : ***le choix non-déterministe*** et ***le choix déterministe***.

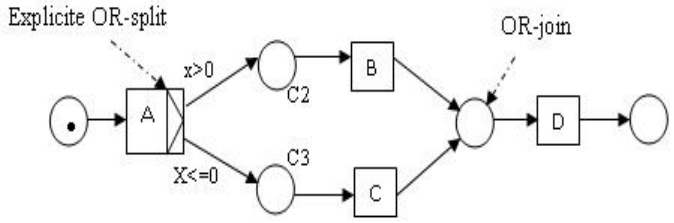
1. **Choix non-déterministe :** Pour modéliser ce comportement deux constructions sont utilisées OR-split et OR-join. La figure suivante représente un choix entre l'exécution des deux tâches B ou C grâce à un jeton dans la place c2, mais cela permet un choix non-déterministe (l'exécution de B ou de C car les deux tâches sont permises).




Routage conditionnel non-déterministe

2. **Choix déterministe :** Le choix entre plusieurs alternatives peut être guidé par les données (ou les attributs) du workflow, dans ce cas c'est un choix déterministe.

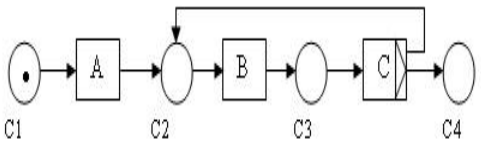
   Nous pouvons prendre le schéma de la figure précédente et ajouter une précondition pour chacune des tâches B et C (en se basant sur un ou plusieurs attributs du workflow).
   Il y a une autre façon de modéliser le choix déterministe comme dans la figure suivante. La transition A comporte deux places de sortie c2 et c3 et produit un seul jeton (soit dans c2 ou dans c3). Le choix entre les deux places est basé sur la valeur de l'attribut x. Un symbole spécial est utilisé pour représenté que la tâche A est un OR-split (exclusive OR).




Choix explicite entre B et C à base de l'attribut x .

**Itération :**

L'itération peut être modélisée en utilisant l'implicite (OR-split, OR-join), l'explicite (OR-split, OR-join). Il est utilisé dans le cas où une ou plusieurs tâches peuvent être exécutées plusieurs fois. La figure suivante montre un exemple d'un modèle de workflow en utilisant un routage itératif.




Itération : B peut être exécutée plusieurs fois

#### 2.2.3.3 Le déclenchement (Triggering)

Le déclencheur (trigger) est une condition externe qui conduit à l'exécution d'une tâche permise. L'exécution d'une instance d'une tâche permise pour un cas spécifique commence au moment où la tâche est déclenchée. Dans ce qui suit on propose

quatre types de déclenchement.

- **Automatique :** La tâche est déclenchée au moment où elle est permise, ce type de déclenchement est utilisé pour les tâches qui sont exécutées par une application qui n'exige pas une interaction humaine.
- **Utilisateur :**Le déclenchement est effectué par un participant humain.
- **Message :** C'est un événement externe qui permet de déclencher une instance d'une tâche permise.
- **L'horloge :** Une instance d'une tâche permise est déclenchée par l'horloge (à un instant donné une instance d'une tâche permise est déclenchée).

La figure suivante schématise les quatre types de déclenchement :

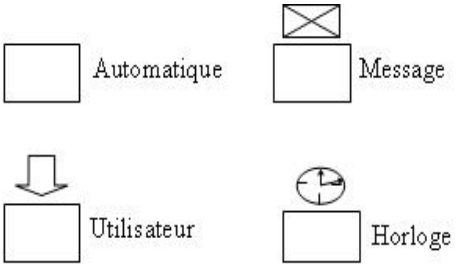



FIG. 2.4: Les quatre types de déclencheurs [3]

#### 2.2.3.4 Exemple d'un réseau workflow (WF-net) :

Dans la figure 2.5 , une définition d'un processus de traitement de plaintes et représentée. Ce processus de workflow est étendu avec les informations de déclenchement. Les tâches Register, Evaluate, Process-complaint, Check-processing requirent une interaction humaine, les tâches Send-questionnaire et archive sont exécutées automatiquement.

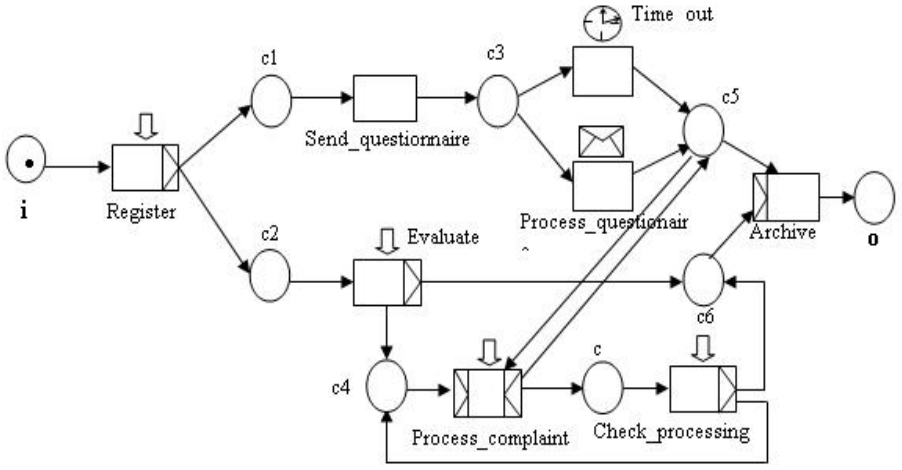



FIG. 2.5: Une définition d'un processus workflow avec une extension par les déclencheurs [3]

### 2.2.4 Conclusion

Dans cette première partie de l'état de l'art, nous avons présenté une approche de modélisation formelle de workflow qui est les réseaux de pétri. Sa sémantique formelle et sa théorie permettent une modélisation graphique a priori et une technique d'analyse rigoureuse qui permet d'être utilisée pour vérifier la conformité de la procédure modélisée.

Néanmoins, l'exécution des tâches de workflow ne peut pas toujours suivre un modèle prédéfini à cause du manque d'informations au moment de la conception. En plus la modélisation a priori de workflow à l'aide des réseaux de pétri ne permet pas la flexibilité et ne donne aucune liberté à l'utilisateur de spécifier ce qu'il veut à cause de leur nature rigide.

La méthode de modélisation déclarative que nous abordons dans la prochaine partie de cet état de l'art vient pour faire face aux problèmes des modèles de réseau de pétri en rendant le modèle plus souple et plus flexible, mais qui reste aussi insuffisante dans le traitement des erreurs d'exécution du modèle.

## 2.3 Approche Déclarative

### 2.3.1 Introduction à l'approche déclarative

L'EXÉCUTION des tâches dans un workflow ne peut pas toujours suivre un modèle de processus prescriptif. Dans les domaines d'applications tel que la gestion de désastre, les workflows sont partiellement spécifiés et les circonstances de leurs exécutions changent.

Il existe plusieurs approches pour la modélisation formelle de WF qui sont efficaces dans telles situations comme les spécifications déclaratives au lieu des spécifications opérationnelles . Ce type de modélisation (déclaratif) permet au concepteur de décrire une scène en utilisant ses caractéristiques et ses propriétés en lui laissant plus de liberté d'organiser son travail, et l'ordinateur serait responsable de l'exploration de l'univers des solutions vraies possibles et de la présentation des scènes qui vérifient la description. Après avoir fourni la description de la scène et la génération automatique faite, le concepteur pourrait choisir l'instance vraie désirée, en utilisant un ensemble d'outils appropriés.

***Les modèles déclaratifs spécifient ce qui devrait être fait sans indiquer comment il devrait être fait*** *[33]* .

L'utilisation de la logique temporelle et plus précisément le langage LTL (Linear Temporal Logic) a été à l'initiative de plusieurs travaux de recherches dans le domaine de la modélisation déclarative du workflow en utilisant des contraintes de la logique temporelle à travers l'ensemble des tâches du workflow. C'est à dire la modélisation déclarative des relations entre les tâches [40] [18].

Dans cette deuxième partie, une discution sur le langage LTL et sa possibilité avec sa syntaxe et sémantique de modéliser un processus (workflow). Une définition des contraintes temporelles et les réseaux de contraintes temporelles pour les processus métiers sera donnée en détail. Nous abordant ainsi la théorie d'Allen pour représenter et raisonner sur l'information temporelle entre les paires de tâches d'un processus.

### 2.3.2 La logique temporelle

Les logiques temporelles permettent de représenter le comportement des systèmes réactifs au moyen de propriétés qui décrivent l'évolution du système. C'est

une extension de la logique conventionnelle (propositionnelle), elle intègre de nouveaux opérateurs qui expriment la notion du temps. En effet, par exemple, il n'est pas possible d'exprimer dans la logique classique une assertion liée au comportement d'un programme telle que *après exécution d'une instruction i, le système se bloque*. Dans cette assertion, les actions s'exécutent suivant un axe de temps : à l'instant **t**, exécution de l'instruction **i**, et à l'instant **(**t+1) blocage du système. Il faut donc une logique qui modélise les expressions du passé et du futur.

#### 2.3.2.1 Langage LTL

La logique temporelle linéaire ou LTL permet de représenter le comportement des systèmes réactifs au moyen des propriétés qui décrivent le système pour lequel le temps se déroule linéairement. En clair, on spécifie le comportement attendu d'un système, en spécifiant l'unique futur possible [**?**].

La figure suivante représente une séquence d'actions qui se suivent dans l'axe du temps :

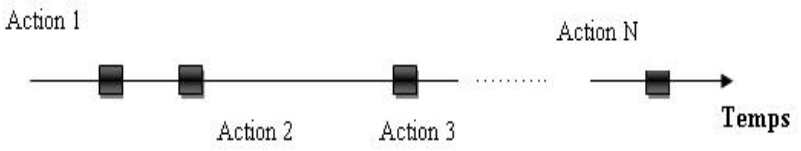



FIG. 2.6: Le déroulement temporel d'un système dans une logique linéaire.

Le langage LTL comporte une syntaxe est une sémantique qui lui permet de spécifier le comportement d'un système réactif (temps réel).

#### 2.3.2.2 Syntaxe de LTL

Les formules de LTL sont définies par la grammaire suivante :

- *Les propositions atomiques :* $\Psi, \Phi : :=$ | p | q | true | false. Avec , $\Psi, \Phi \in$ LTL
- *Les connecteurs booléens :* Soit $\Psi, \Phi \in$ LTL.
  Toute formule de la forme : $\psi \mid \psi \wedge \phi \mid \psi \vee \phi \mid \psi \Rightarrow \phi \mid \psi \Leftrightarrow \phi \mid \in$ LTL.
- *Les connecteurs temporels :* Soit $\Psi, \Phi \in$ LTL. Toute formule de la forme : $G\Psi \mid F\Psi \mid \Psi \cup \Phi \mid X\psi \quad \in$ LTL.

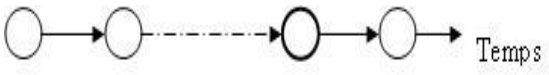
On peut les trouver aussi avec la notation suivante :

G = ❑ (Toujours).
F = ♦ (Eventuellement).
U = U (Jusqu'à).
X = ❍ (Suivant).

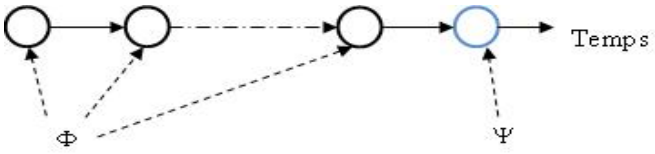
#### 2.3.2.3 Sémantique de LTL

On peut résumer la sémantique de la logique temporelle LTL comme suit : Notons que les ronds renforcés noirs expriment que Φ est vraie.

- X (Next *suivant* ) : X Φ : Φ est vérifié à l'état suivant




- F (Eventually , *éventuellement* ) : F Φ : il existe un état pour lequel Φ est vraie .
- G (always, *toujours* ) : G Φ : Φ est toujours vérifiée dans le futur.
- G F p : p est vérifiée une infinité de fois dans le futur.
- F G p :p est toujours vérifiée à partir d'un certain état.
  Dans ce cas le rond renforcé bleu exprime que Ψ est vraie.




- U (Until, *jusqu'à*) Φ *U* Ψ : Φ est vérifiée jusqu'à ce que Ψ le soit.

#### 2.3.2.4 Formules LTL

Les formules LTL permettent dans notre cas (spécification de workflow) de représenter un ensemble de contraintes, chaque séquence ordonnée d'un ensemble d'occurrence de tâches de workflow qui satisfait de telles formules représente un chemin exécutable dans le workflow. ***Ceci signifie que l'exécution finale du workflow impose la satisfaction de toutes ses contraintes***.

Un chemin est comme définit par Fahland [19], une séquence infinie d'interprétations de propositions r : $N \rightarrow 2^{prop}$ . Chaque proposition $i \in N$ dans **r** est conçue comme un état.

Une formule LTL notée $\Phi \in F_{LTL}$(Prop) est évaluée comme un état $i \in N$ dans le chemin r.

- (r, i)$\models p \in Prop$ Si et seulement Si $p \in r(i)$.
- (r, i)$\models$ ❍ $\Phi$ Si et seulement Si (r, i+1)$\models \Phi$.
- (r, i)$\models$ ❑ $\Phi$ Si et seulement Si $\forall\ (j \geq i)$ (r,i)$\models \Phi$.
- (r, i)$\models$ ♦ $\Phi$ Si et seulement Si $\exists (k \geq i)$ (r,k)$\models \Phi$.
- (r, i)$\models$ [ $\Phi$ U $\Psi$ ] Si et seulement Si $(k \geq i)$ (r,k)$\models \Psi \wedge \forall\ i \leq j < k$ (r,j)$\models \Phi$.

### 2.3.3 LTL pour les workflows

Dirk Fahland [19], Aalst et Pesic ont choisi un sous ensemble spécial de LTL pour représenter les workflows. Ils ont définit un workflow déclaratif D=(tâ$che_D$ , $\Phi_D$) qui est composé de :

- Un ensemble de tâches atomiques qui peuvent être exécutées dans D (tâ$ches_D$).
- Une formule LTL $\Phi_D \in F_{LTL}$ (Prop(D)). Avec $F_LTL$ (Prop(D)) comme un ensemble de propositions représentées avec le langage LTL (ensemble de contraintes) [19].

L'ensemble de propositions atomiques Prop(D) a une forme spéciale : Prop(D)=df$\Big\{$activite =A ,A $\in$(tâ$ches_D$) $\Big\}$. Si *activité* $= A$ est une activité prise dans un état, cela signifie que cet état est le représentant de cette tâche.

Cependant, Un chemin r : N $\rightarrow 2^{(}Prop$) est faisable dans D Si $\forall$ i, | r(i)| $\leq 1$. Le comportement spécifié par D est l'ensemble des exécutions possibles satisfaisant la formule ($\Phi_D$). Ce comportement est interprété par R(D) df $\Big\{r \models \Phi_D, r\ est faisable\Big\}$ c.à.d. ***l'ensemble de chemins de tâches satisfaisantes*** $\Phi_D$.

Aalst et Pesic ont défini un ensemble de modèles de contraintes LTL ($F_{WF}$) comme étant des blocs de construction de base d'un langage déclaratif de services de flux **(DecSerFluw)**. C'est un Workflow déclaratif D où $\Big\{\Phi_D = \wedge_{i=1}^{n} \Phi_i\Big\}$ qui est une conjonction de contraintes du workflow $\Phi_i \in F_{WF}(D)$ , $i = 1 \ldots$ n. Chaque contrainte possède une représentation graphique qui permet une spécification déclarative sans l'utilisation directe de formule LTL.

Un exemple d'une modélisation d'un processus avec le langage LTL avec une explication est donné dans l'annexe C.

### 2.3.4 Le système de contraintes pour l'exécution d'un processus métier

Dans ce qui suit, un cadre de modélisation d'un processus métier sera présenté qui favorise le besoin d'une flexibilité dans l'exécution. Cette modélisation est basée sur la notion des contraintes du processus.

#### 2.3.4.1 Les contraintes du processus

Les contraintes d'un processus peuvent être définies pour plusieurs cas, comme la sélection des tâches, l'allocation de ressources, le contrôle de flux ...etc.

Dans le cas de la modélisation d'un processus métier ou la spécification de workflow, nous nous intéressons aux contraintes d'ordonnancement des tâches et leurs représentation sous forme d'un réseau de contraintes temporelles afin d'avoir des chemins d'exécutions faisables pour le workflow en question en satisfaisant l'ensemble de ses contraintes.

#### 2.3.4.2 Les classes de contraintes

Il existe trois classes de contraintes qui expriment les contraintes sous lesquelles un processus métier peut être exécuté de tel façon que les buts commerciaux peuvent être efficacement atteints [27].

- *Contraintes de sélection (selection constraints)* : Contraintes qui définissent quelles activités constituent le processus.
- *Contraintes d'ordonnancement (scheduling constraints)* : Elles définissent quand les activités doivent être exécutées.
- *Contraintes de ressources (resource constraints)* : Elles définissent quelles ressources sont requises pour l'exécution des activités.

#### 2.3.4.3 Les niveaux d'application des contraintes

Ces trois classes de contraintes sont applicables à deux niveaux différents.

- *Niveau tâches :* Les contraintes au niveau tâches constitue la spécification de plusieurs propriétés d'une tâche individuelle dans le processus, incluant les ressources de l'activité (application, rôle, les exécuteurs, données ...etc.) ainsi que le temps (les contraintes de durée et la date de fin).
- *Niveau processus :* Les contraintes de niveau processus spécifient quelles activités doivent être incluses dans le processus, ainsi que les dépendances de flux

à travers ces activités et aussi les dépendances de contrôle (tel que l'ordre, l'alternative, le parallèle ...etc.) et les dépendances temporelles d'interactivités (date limite relative).

### 2.3.5 La satisfaction de contraintes et réseau de contraintes

La satisfaction de contraintes est une approche bien connue de résolution de contraintes où un problème est formulé sous forme d'un **C**onstraint **S**atisfaction **P**roblem (CSP). La modélisation de contraintes est le processus de formulation du problème, et le traitement de contraintes est le processus de raisonnement sur ces contraintes ainsi que la recherche de solution.

La représentation d'un problème sous forme d'un réseau de contraintes nous permet de représenter graphiquement les relations existantes entre l'ensemble de ses contraintes ***(dans notre cas entre l'ensemble des tâches du processus)***.

Soit N le réseau de contraintes qui nous permet de modéliser et de représenter un problème donné, il est composé du triplet < X, D, C > tel que :

- X est l'ensemble fini de variables $\left\{X1, X2, X3, \ldots Xn\right\}$ avec les Domaines D respectifs.
- D est le domaine des variables, il contient les valeurs possibles pour chaque variable.
- C est l'ensemble de contraintes tel que C=$\left\{C_1, C_2, \ldots, C_m\right\}$, où chaque $C_i$ est une relation qui impose une limitation aux valeurs d'une variable, elle peut être aussi une combinaison de variables.

#### 2.3.5.1 Exemple d'une modélisation par un réseau de contraintes

Voici un exemple simple d'un problème représenté comme l'assignation d'une suite de valeurs $\left\{x, y, z\right\}$ d'un domaine D= $\left\{1, 2, 3\right\}$ tel que x>y et y>z. On peut représenter ce problème sous forme d'un CSP < X, D, C> avec :

- X=$\left\{x, y, z\right\}$.
- D=$\left\{Dx, Dy, Dz\right\}$, Dx=Dy=Dz=$\left\{1, 2, 3\right\}$.
- C=$\left\{C_{xy}, C_{yz}\right\}$ $avec\, C_{xy} = (x > y)\, et\, C_{yz} = (y > z)$.

La représentation de ce problème sous forme d'un graphe de contraintes (réseau de contraintes) où les sommets représentent les variables et les arcs représentent l'existence d'une contrainte entre deux variables.

La figure suivante représente le graphe de contraintes du problème précédent :

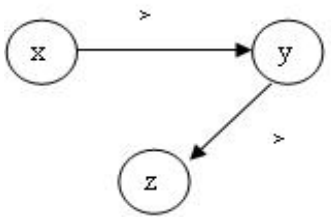



FIG. 2.7: Graph de contraintes .

### 2.3.6 Les contraintes d'ordonnancement

Tous les types de contraintes déjà mentionnées sont inter reliées mais celles qui nous intéressent pour la spécification de processus (workflow) sont les contraintes de niveau processus et plus précisément les contraintes d'ordonnancement. Ce type de contraintes indique les séquences possibles d'activités qui peuvent s'exécuter dans une spécification de processus en satisfaisant un certain contrôle de flux.

Des conditions pour représenter le raisonnement sur les contraintes d'ordonnancement d'un processus métier nécessitent un cadre plus formel pour capter les contraintes temporelles entre les activités du processus ; une telle information temporelle est souvent indéfinie et inachevée.

### 2.3.7 Relations d'intervalle d'Allen

Pour représenter ce type d'information temporelle dans un processus métier, treize relations de base entre deux intervalles sont données par Allen [9]. Par exemple la relation $\{avant, rencontre\}$ ou (before, meets) entre les intervalles X et Y, spécifie que X fini avant Y ne commence ou X fini juste avant Y. Voici un ensemble de relations d'intervalles selon Allen .

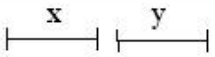
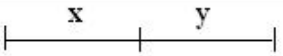
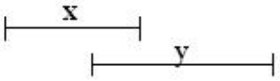
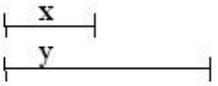
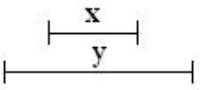
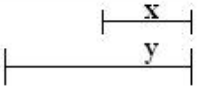
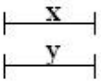
| Relation | Symbole | Inverse | Signification |
|---|---|---|---|
| x avant y (x before y) | **b** | $\mathbf{b_i}$ |   |
| x rencontre y (x meets y) | **m** | $\mathbf{m_i}$ |   |
| x chevauche y(x overlaps y ) | **o** | $\mathbf{o_i}$ |   |
| x commence y (x starts y) | **s** | $\mathbf{s_i}$ |   |
| x durant y (x during y) | **d** | $\mathbf{d_i}$ |   |
| x fini y (x finishes y ) | **f** | $\mathbf{f_i}$ |   |
| x égale y (x equals y) | **eq** | $\mathbf{eq_i}$ |   |

FIG. 2.8: Les relations d'intervalle selon Allen [27].

Il existe d'autres relations possibles qu'on peut définir entre deux intervalles ; tels que *l'inverse, l'intersection et la composition*, elles sont définies comme suit :

- **Inverse** : L'inverse $R^{\cup}$ d'une relation R est la relation $R^{\cup}= \Big\{(a,b)|(b,a) \in R\Big\}$
- **Intersection** : L'intersection de deux relation R et R' de IA notée par R $\cap$ R' est l'ensemble d'intersection théorique de R et R'.
  Par exemple : R = $\Big\{o, s, f, m\Big\}$ et R'=$\Big\{s, f, d\Big\}$, R $\cap$ R'=$\Big\{s, f\Big\}$
- **Composition** : La Composition de deux relations d'intervalle de base r et r' notée pas r $\otimes$ r' est identifiée dans le tableau suivant.

| **r \ r'** | **b** | **s** | **d** | **o** | **m** |
|---|---|---|---|---|---|
| **b** | b | b | b o m d s | b | b |
| **s** | b | s | d | b o m | b |
| **d** | b | d | d | b o m d s | b |
| **o** | b | o | o d s | b o m | b |
| **m** | b | m | o d s | b | b |

TAB. 2.1: Une portion de la table transitive défini par Allen [27].

### 2.3.8 Les réseaux de contraintes temporelles

Les réseaux de contraintes temporelles (**T**emporal **C**onstraint **N**etworks (TCN)) sont une sous classe de réseaux de contraintes où la représentation des informations temporelles peut être vue comme un réseau de contraintes binaires. Des techniques de satisfaction de contraintes peuvent être utilisées pour raisonner sur ce type d'informations.

Allen a proposé un cadre formel pour représenter et raisonner sur les contraintes temporelles en un réseau IA (Interval Algebra) qui se base sur la définition de treize relations d'intervalle. Une classe restreinte de réseau IA peut être traduite en un réseau PA (Point Algebra) dans un temps polynomial qui permet aucune perte d'information.

#### 2.3.8.1 Le réseau PA (Point Algebra)

Le réseau PA (**P**oint **A**lgebra (PA) ou algèbre de points) est un réseau de relations binaires où les variables représentent des points dans le temps et les relations binaires entre les variables sont les disjonctions de relations de points d'intervalles.

Les relations permises entre deux intervalles dans PA sont un sous ensemble IA. Ces relations peuvent être représentées en utilisant les relations ($<$, $=$, $>$) dans des conjonctions de relations entre les points finaux des intervalles (C'est-à-dire des relations entre les débuts et les fins des intervalles) [27].

### 2.3.9 Réseau de contraintes temporelles pour les processus métiers (BPCN)

Pour raisonner sur les informations temporelles entre les tâches d'un processus Selon [27]. Allen a représenté une tâche T comme étant un intervalle de temps $(T^-, T^+)$ tel que $T^- < T^+ \; où \; T^- et \; T^+$ sont interprétés comme des points dans le temps linéaire. $T^-$ est le point de début de l'exécution de la tâche T, et $T^+$ est le point de fin de son exécution.

- **Exemple** :

Soit les intervalles $(T1^-, T1^+) et (T2^-, T2^+)$ dénotant le début et la fin des tâches T1 et T2 respectivement. Supposant qu'il existe une relation entre T1 et T2 qu'on peut représenter en IA comme étant T1$\Big\{meets\Big\}$T2. La représentation de cette re-

lation en PA sera : $(T1^- < T2^-)(T1^- < T2^+)(T1^+ = T2^-)(T1^+ < T2^+)$.
On peut représenter schématiquement ce comportement comme suit :

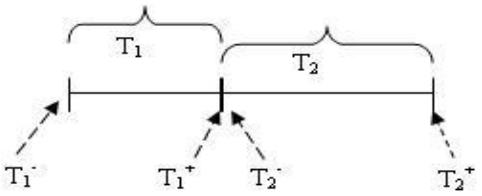



FIG. 2.9: Schéma de la relation T1 meets T2.

Avec cette représentation des points délimitant le début et la fin d'une tâche, on peut raisonner sur cette information et représenter une ou plusieurs relations temporelles entre chaque paire de tâches **(une relation au moins est exigée pour chaque paire de tâches)**, ainsi de les formaliser sous forme de réseau de contraintes temporelles et raisonner sur les contraintes d'ordonnancement.

Dans le cas de l'existence d'une seule relation entre chaque paire de tâches on aura un ordre total entre les tâches, et dans le cas contraire un ordre partiel est indiqué.

- ***Une instance d'un processus est une instance totalement ordonnée si pour chaque paire de tâches une seule des treize relations est utilisée***.
- ***Un ordre partiel correspond à certaines relations entre les tâches qui tiennent compte d'un grand nombre de possibilités dans lesquelles il existe plusieurs exécutions d'instances de processus*** [27].

#### 2.3.9.1 Formulation d'un BPCN

Un **B**usiness **P**rocess **C**onstraint **N**etworks (BPCN) est un réseau de contraintes temporelles N= <X, D, C> où :

- X= $\left\{T_1, T_2, \ldots, T_N\right\}$ est l'ensemble de toutes les tâches du processus métier représentées avec des intervalles de temps.
- D = $\left\{D_1, D_2, \ldots, D_N\right\}$ est l'ensemble de domaines de paires ordonnées de valeurs de temps discret, par exemple $\left\{(s,e)|s<e\right\}$ cela correspond aux points de début (s) et de fin (e) dans l'intervalle d'une tâche T.
- C= $\left\{C_{ij}\right\}$ c'est un ensemble de contraintes $C_{ij}$ qui permet de définir l'existence d'une contrainte entre les deux tâches $T_i$ et $T_j$. Elle est définit comme $C_{ij} \subseteq$

$\left\{b, b_i, m, m_i, o, o_i, s, s_i, f, f_i, eq\right\}$ (voir la figure 2.8). Ces contraintes décrivent les points de localisation permis des tâches dans le temps linéaire.

#### 2.3.9.2 Résolution de BPCN

Une solution d'un BPCN est l'attribution d'une paire de valeurs pour chaque variable (tâche) de tel sorte qu'aucune contrainte (temporelle) ne soit violée. Une solution peut être établit en assignant une seule relation pour chaque paire de tâches (afin d'éviter l'ambigüité), cela correspond à un ordre total du modèle d'exécution de processus.

#### 2.3.9.3 Consistance d'un BPCN

La consistance est utilisée pour décrire la qualité des contraintes définies dans le réseau de contraintes, si des conflits existent entre les contraintes, alors on peut conclure que le réseau de contraintes est inconsistant et par conséquent aucune solution n'existe.

Dans un BPCN donné, il peut y avoir plusieurs instances de processus qui peuvent satisfaire les exigences d'un processus métier, mais il faut au moins l'existence d'une instance satisfiable.

Puisque les treize relations d'intervalles sont totalement ordonnées, le réseau IA d'Allen nous permet de définir plusieurs relations sur la même paire de variables. Cependant, des conflits de contraintes dans BPCN existent seulement pour les paires de variables différentes [27] .

On défini le réseau N' comme un scénario consistant du réseau N si et seulement si :

Il existe exactement une seule relation entre chaque paire de variables $(T_i, T_j)$ dans N' notée $|R'_{ij}|=1$ et.

1. Chaque relation R' dans N' est un sous ensemble entre la même paire de variables dans N. notée $R'_{ij} \subseteq R_{ij}$.
2. Il existe une instanciation consistante de N'.

Alors pour trouver un scénario consistant on cherche simplement les réseaux satisfaisants les conditions 1 et 2 [27]

Un exemple d'un réseau inconsistant est donné dans l'annexe B.

#### 2.3.9.4 Un chemin consistant dans BPCN

L'exécution d'un processus métier nécessite l'exécution d'un ensemble de tâches ordonnées selon la définition du processus ou la spécification du workflow, cela correspond à trouver un scénario consistant pour le réseau de contraintes qui permet de contrôler l'exécution de ce processus. Si on schématise ce comportement on peut le voir comme l'existence d'un **chemin consistant** dans le réseau de contraintes temporelles du processus en question [27].

#### 2.3.9.5 L'algorithme d'un chemin consistant

Une contrainte binaire $C_{ij}$ est un chemin consistant pour la variable $T_k$ si et seulement si : $(C_{ij} \cap (C_{ik} \otimes C_{kj})) \neq \emptyset$.

Un BPCN est un chemin consistant si et seulement si :

$\forall$ **R (incluant les relations universelles) et** $\forall k \neq i, j$**.R est un chemin consistant pour** $T_k$ .

**Algorithm 2.1** L'algorithme de chemin consistant [27]

```
1: Input : An IA network N
2: Output : A path-consistent IA network
3: Début
4: pour k=1,..., n faire
5:    pour i,j=1,..., n  faire
6:       begin
7:       (C_ij ← (C_ij ∩ (C_ik ⊗ C_kj))
8:       si C_ij= ∅ alors
9:          break.
10:      finsi
11:   fin pour
12: fin pour
13: Fin.
```

Cet algorithme permet de valider la définition des contraintes dans un réseau BPCN. L'application à plusieurs reprises de l'algorithme au réseau BPCN jusqu'à ce qu'au moins une contrainte devienne vide, indique l'existence d'un conflit. Par conséquent aucune solution n'existe pour ce réseau.

### 2.3.10 DecSerFlow (Declarative Service Flow Language)

#### 2.3.10.1 Définition

La difficulté de la lecture et la compréhension des langages déclaratifs comme LTL a conduit au développement de plusieurs langages graphiques (DecSerFlow [40], Declare [31], Condec [33]) pour aider l'utilisateur dans sa tâche de modélisation de processus.

DecSerFlow comme son nom l'indique est un langage déclaratif graphique de modélisation de service de flux. Il permet de définir une syntaxe graphique étendue de quelques contraintes de type LTL incluses dans un service de flux. La combinaison de cette syntaxe graphique et sa représentation en langage LTL forme le langage DecSerFlow [40].

#### 2.3.10.2 Pourquoi DecSerFlow

L'utilisation d'un style déclaratif dans la spécification de processus permet de décrire exactement ce qui est nécessaire sans forcer le concepteur à ajouter des activités ou des conditions supplémentaires comme dans le cas des langages procéduraux. Par exemple si on a deux activités A et B qui s'exécutent plusieurs fois l'une exclut l'autre, avec un langage procédural, il est difficile de spécifier un tel comportement sans ajouter des conditions et des contraintes supplémentaires.

Pour l'exemple précédent, avec un langage procédural un choix doit être effectué entre l'exécution de A ou de B, et le nombre d'exécutions de l'une des activités doit être indiqué. Par contre, l'utilisation d'un style (langage) déclaratif peut éviter ce type de spécification. Dans le langage LTL (Linear Temporal Logic) la formule $\neg$ ( $\blacklozenge$A $\wedge$ $\blacklozenge$B) permet suffisamment de spécifier de tel comportement, malheureusement les langages déclaratifs (comme LTL) sont difficiles à comprendre et à utiliser par les non experts, c'est pour cette raison que le développement des langages graphiques tel que DecSerFlow basé sur LTL permet une spécification facile des processus avec un style déclaratif.

La figure suivante représente une vue de l'utilisation de DecSerFlow [40] , la partie gauche (design-time) de la figure montre un processus avec quatre activités A B C et D et trois contraintes entre elles.

- Une connexion entre A et C signifie que n'importe quel occurrence de A devrait par la suite être suivi par au moins une occurrence de C (C'est-à-dire $\square(A \rightarrow \blacklozenge C)$ en formule LTL).

- Une connexion entre A et B signifie que on ne peut pas avoir le cas où A est exécutée et B est exécutée, elle est décrite en LTL comme ¬ (♦A ∧ ♦ B)
- Une dernière contrainte connectant D et B décrit que pour chaque occurrence de D implique l'occurrence de B avant ou après, cela est représente par la formule LTL : (♦ D → ♦ B), les séquences suivantes [D,D,D,B], [B,D,D] sont des séquence permises.

La partie du milieu (mapping) permet de traduire les représentations graphiques en formules LTL. Et enfin la partie droite (run-time) représente le processus en temps d'exécution.

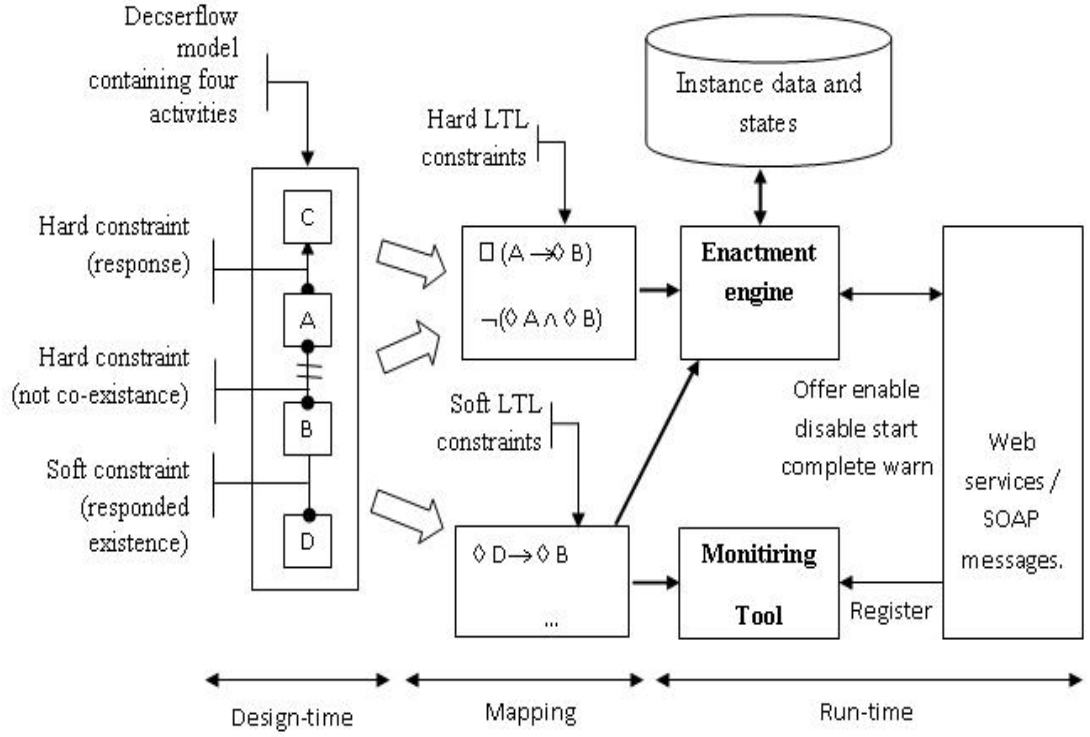



FIG. 2.10: Overview of the role played by DecSerFlow in supporting Services flows [40].

#### 2.3.10.3 Créer un modèle avec DecSerFlow

Dans ce qui suit on va présenter les étapes nécessaires pour modéliser un service de flux à l'aide de DecSerFlow.

1. **Création d'activités :** La notion d'activité est semblable dans n'importe quel langage de workflow. C'est une activité atomique, elle correspond à une unité logique de production. Les relations entre ces activités sont représentées

dans DecSerFlow comme des contraintes. Chacune représente une règle dans le processus métier.

2. **Création de contraintes :** Une contrainte indique une politique (ou un règle économique) entre les activités du modèle. A n'importe quel moment de l'exécution du service une contrainte est évaluée à vrai ou à faux, cette valeur peut changer pendant l'exécution.

   L'exécution du service est correcte à un certain point du temps si l'évaluation de toutes les contraintes est à vrai. Par exemple si nous prenons la formule ❑ (A → ♦ B), tel que A et B sont des activités, cette formule signifie que à chaque exécution de A une exécution de B est suivis.
   Initialement l'expression est évaluée à vrai, après l'exécution de A, l'expression est évaluée à faux et elle reste à faux jusqu'à ce que B s'exécute.

   **Le but :** *C'est de terminer l'exécution du service dans un état où toutes les contraintes sont évaluées à vrai.*

Pour créer des contraintes dans DecSerFlow nous employons ce qu'on appelle constraint templates. Chaque template se compose d'une formule LTL et une représentation graphique de cette formule.

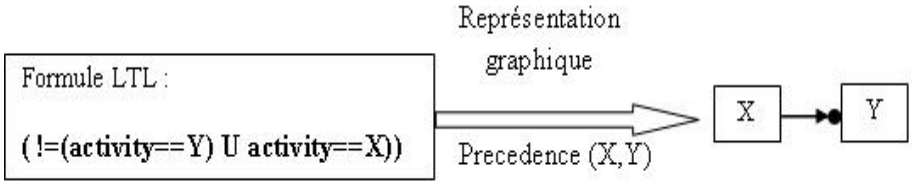



FIG. 2.11: Une exemple du contrainte Template.

Les dépendances ou les contraintes entre les activités sont reformulées sous forme de formules LTL qui sont à leur tour représentées graphiquement, ce qui nous donne un ensemble de *constraint templates* initial ou en d'autre terme : chaque *constraint template* peut être utilisé afin de spécifier des contraintes sur les activités dans plusieurs modèles de DecSerFlow.

Dans l'ensemble initial de *constraint templates* on distingue trois types de templates :

1. **Existence** : Ce type de formule représente le nombre possible d'exécutions d'une tâche ou la cardinalité d'une tâche.
2. **Relation** : Elle défini les relations ou les dépendances entre deux tâches.
3. **Négation** : Sont les négations des formules de relations.

La figure suivante représente un exemple de ces trois types de Templates [40].

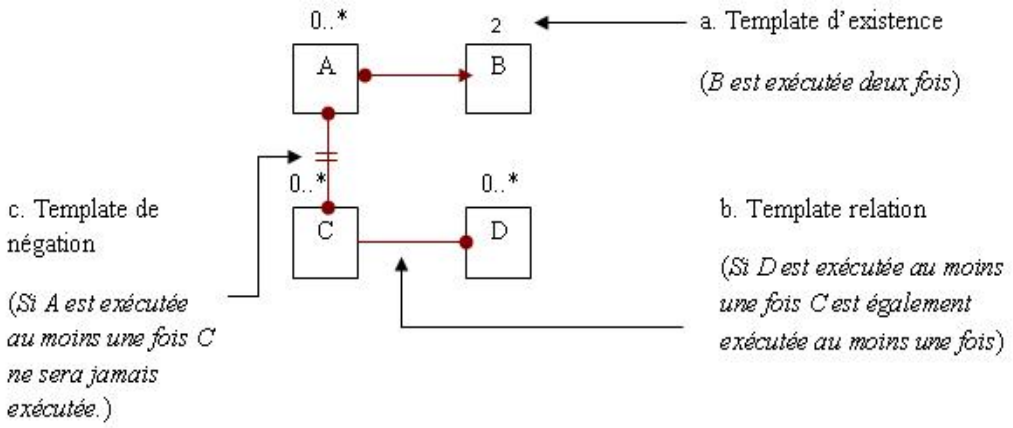



FIG. 2.12: Un modèle DecSerFlow représente un exemple de notation [40].

Un exemple détaillé d'une modélisation d'un service par DecSerFlow est donné dans l'annexe C.

### 2.3.11 Conclusion

Nous avons présenté dans cette deuxième partie de l'état de l'art un cadre déclaratif de modélisation d'un processus métier, le rôle primordial que joue le langage LTL dans cette modélisation. L'apport de l'approche de réseaux de contraintes temporelles dans la recherche de chemin consistant et ainsi la (les) solution(s) pour le processus métier.

Malgré tous les avantages qu'elle offre à l'utilisateur dans sa liberté de spécification. La modèlisation déclarative n'est pas parfaite, il existe toujours certaines lacunes qui persistent concernant sa difficulté de modéliser des scènes très complexes,

sa gourmandise en ressources d'ordinateur et l'exploitation d'un grand nombre de scènes,**et le plus important est le manque de contrôle dans les spécifications ce qui entraîne plusieurs erreurs lors de l'exécution**.

# Chapitre 3

# La flexibilité pour les workflow

## 3.1 Introduction

LA flexibilité est un axe de développement important dans le domaine de la gestion de workflow. Une façon de la prendre en compte est de permettre une modélisation partielle plutôt que d'imposer un modèle impératif complètement défini.

Dans ce présent chapitre nous étudions la flexibilité pour la spécification de workflow, la liberté qu'elle offre à l'utilisateur pour spécifier ce qu'il veut et comment l'incorporer dans une spécification à l'aide du mécanisme de poche de flexibilité.

## 3.2 Définition de flexibilité

Elle est définit comme étant ***la capacité d'un processus workflow d'exécuter avec un modèle partiellement spécifié, où la spécification finale est faite au moment de l'exécution et qui peut être unique à chaque instance*** [29] [30].

Des exemples de processus qui permettent cette flexibilité peuvent être trouvés dans plusieurs applications. Un exemple typique est les soins médicaux, où les procédures d'admission des patients sont prévisibles et répétitives. Cependant les traitements des hospitalisés sont prescrits uniquement pour chaque cas mais ils doivent être contrôlés et coordonnés.

Un modèle flexible est défini alors d'une façon plus étendue ou plus flexible

ce qui permet aux différentes instances de définir leur propre chemin d'exécution (processus).

## 3.3 Cadre de spécification de workflow flexible

Introduire la flexibilité dans les spécifications de workflow avec les langages de modélisation impératifs qui ont des origines dans les formalismes de réseaux de pétri (ou processus algébrique) nous conduit à l'ajout de constructions supplémentaires afin de capter tous les possibilités et alternatives d'exécution. Ceci peut conduire rapidement à des erreurs et surtout rend la spécification difficile à définir et à vérifier.

Cependant, on trouve dans la littérature ( [37] [36]) une approche qui facilite la spécification flexible de workflow en évitant les erreurs et les faiblesses qui peuvent survenir lors de l'ajout des constructions dans les langages impératifs. Ce cadre de spécification est basé sur les **poches de flexibilités (pockets of flexibility)** que nous allons étudier en détails dans ce qui suit.

## 3.4 Modèle de spécification de workflow flexible

La flexibilité dans la spécification de workflow est une tâche primordiale qui permet de capter le maximum d'instances d'exécution et satisfaire les entreprises afin de faire face au changement continu. Cette modélisation doit être **partielle** si on veut satisfaire notre définition de la flexibilité donnée dans la section (3.2). Cette modélisation comporte la définition du ***noyau du processus*** qu'on nomme ***workflow flexible***.

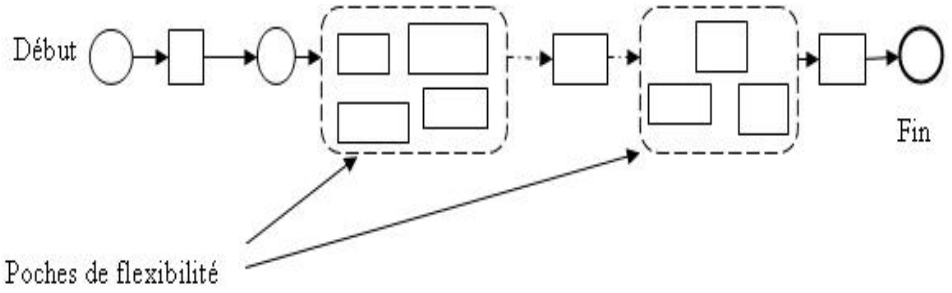
## 3.5 Définition d'un workflow flexible

Selon Shazia Sadiq et al [37], un workflow flexible ou le noyau du processus consiste en deux points :

- Définir des activités et les dépendances de contrôle de flux que comporte le workflow formant ainsi le **noyau du processus**.
- Définir une ou plusieurs poches de flexibilité (pockets of flexibility) dans le processus. Elles sont représentées par des activités spéciales appelées **les activités de construction (build activity)**. Une poche de flexibilité comporte :

– **Un ensemble de fragments** : Où un fragment peut être une activité élémentaire ou un sous processus (ensemble d'activités).
– **Un ensemble de contraintes** : Permet de caractériser une poche à travers une composition valide de fragments de workflow.

Le schéma suivant représente un exemple d'un modèle de workflow qui comporte deux poches de flexibilité.




Exemple d'un workflow flexible.

### 3.5.1 Les poches de flexibilité

Le contrôle de flux entre les fragments de workflow ne peut pas être complètement prédéfini dans le noyau du processus. Le mécanisme de poche de flexibilité permet de combler cette lacune et de spécifier complètement le processus.

#### 3.5.1.1 Définition d'une poche de flexibilité

C'est un mécanisme qui permet d'incorporer une certaine flexibilité dans la définition d'un processus workflow. Elle est composée de :

1. Un ensemble de fragments de workflow (activité élémentaire ou un sous processus).
2. Un ensemble de contraintes que le concepteur peut définir dans le temps de conception. Les contraintes définies permettent de contrôler la validation de la composition de la poche.

La figure suivante représente les composants d'une poche de flexibilité.

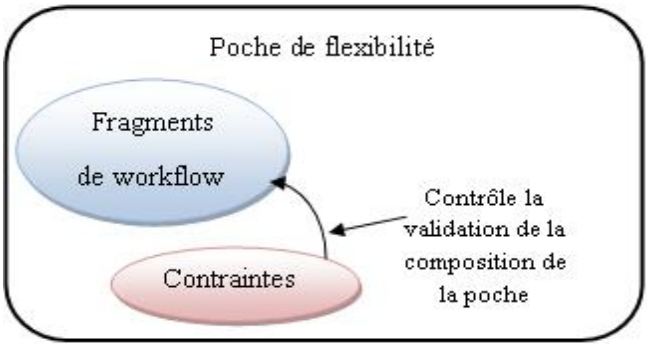



FIG. 3.1: Composants d'une poche de flexibilité.

#### 3.5.1.2 Exemple

La figure 3.2 représente un exemple concret d'un workflow flexible (noyau du processus) qui comporte une seule poche de flexibilité. C'est un procédé simplifié de CRM (gestion des appels clients). Le processus est constitué de plusieurs étapes :

**1.** Après la réception de l'appel du client, un agent du centre d'appel créera dans le système une requête avec ses détails appropriés et l'assigne à un ingénieur qualifié (Enregistrer la requête du client).

**2.** Quand la requête sera dans la liste de travail de l'ingénieur, il devra maintenant prendre une décision sur la façon de résoudre la requête (Résoudre le problème). Si aucune circonstance n'est apparue, le problème est résolu selon une procédure prédéterminée .

**3.** Dans le cas contraire un niveau plus élevé est exigé (Assigné le support de niveau 2). A cause de la difficulté de trouver des réponses exactes aux requêtes du client, une certaine flexibilité doit être offerte à l'ingénieur. Cette flexibilité est fournit dans **la poche de flexibilité (Build task)** mais sous un contrôle donné (l'ordre d'exécution, le nombre de fois d'exécution de chaque activité ...etc.). Par exemple pour la poche de la figure 3.2 n'importe quel test peut être effectué mais un seul à la fois.

**4.** Après l'accomplissement des activités de la poche, des documents résultats seront fourni, et enfin une réponse doit être remise au client pour l'informer du résultat de son appel.

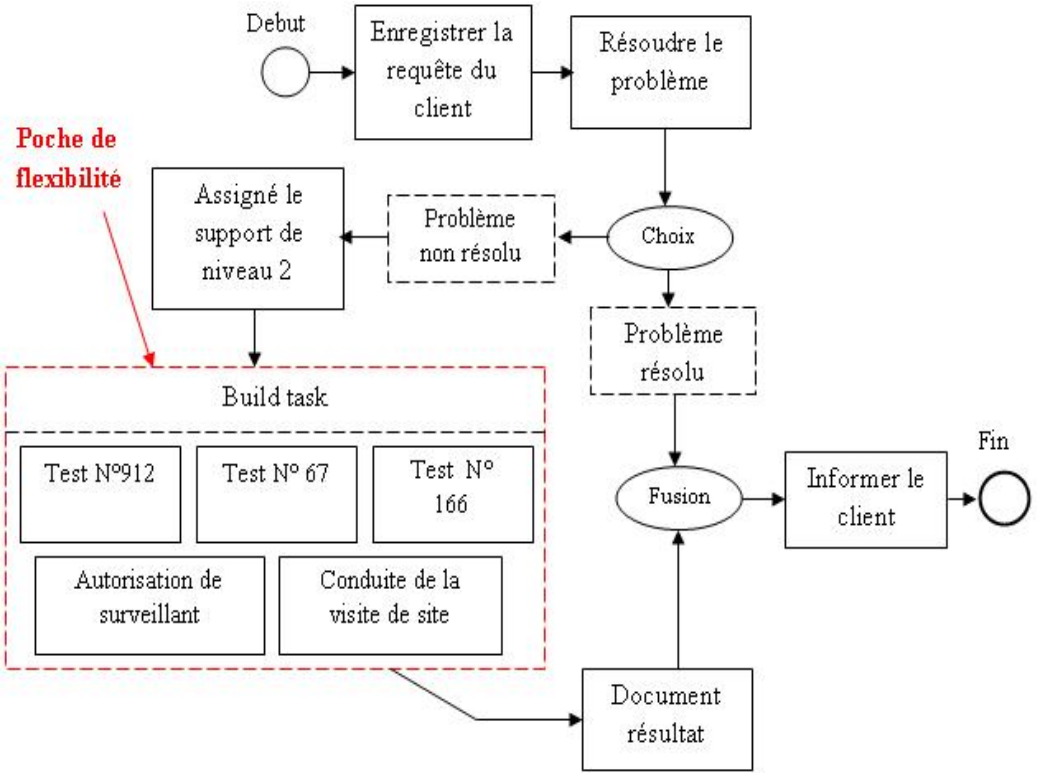



FIG. 3.2: Un workflow flexible [37].

### 3.5.2 Les composants d'un workflow flexible

Dans la section précédente, nous avons défini un workflow flexible comme étant un ensemble de composants : le noyau de processus (avec des activités simples et les activités spéciales représentant les poches de flexibilité), les fragments, et les contraintes de construction. Les sections suivantes détaillent ces trois composants et discutent la façon dont les spécifications de ces composants peuvent être accordées pour fournir un équilibre entre la flexibilité et le contrôle.

#### 3.5.2.1 Le noyau du processus

Dans le noyau du processus une ou plusieurs poches sont construites à partir des fragments et des contraintes. Une fois construite, une poche est fondamentalement un sous processus dans le processus parent, la seule différence est que la poche est une version plus restreinte du langage prescriptif utilisé pour spécifier le noyau du processus. Cependant, construire une poche à partir des fragments et contraintes données ou construire un processus à partir des fragments et contraintes données est conceptuellement semblable.

#### 3.5.2.2 Les fragments

La construction d'une poche à partir de fragments prédéfinis sera plus réaliste parce qu'il fournit un certain degré de contrôle sur ce qui est inclut dans l'instance. Cependant, de nouveaux fragments peuvent être défini pour des instances particulières, et peuvent être réutilisés plus tard dans d'autres instances.

#### 3.5.2.3 Les contraintes de construction

Les spécifications de contraintes dans la composition des fragments est une approche nouvelle dans les spécifications de workflow. La construction des instances peut être contraint par plusieurs facteurs comme : les données spécifiques de chaque instance, l'étape d'exécution de l'instance, les contraintes temporelles, fragments disponibles et les règles métiers de l'application pour laquelle le Template est défini.

Contrairement aux contraintes fortes qui imposent un ordre d'exécution entre les activités (imposer par les modèles impératifs), les contraintes de construction permettent une flexibilité dans les spécifications et ainsi avoir un très grand nombre d'instances.

## 3.6 Spécification d'instances pour le workflow flexible

La spécification d'instances consiste initialement d'une copie de noyau du processus. Pendant l'exécution une instance particulière ce produit et les activités de construction (*build task*) fournissent les moyens d'adapter le noyau du processus à cette instance particulière. La spécification d'instance avant la construction est une **Instance Ouverte** et après la construction c'est une **Instance Template(modèle d'instance)**.

### 3.6.1 Instance Template

L'instance Template est une composition particulière de fragments dans un workflow flexible. Elle représente un modèle de processus pour une instance particulière.

L'exécution du modèle aura lieu avec la pleine application de toutes les contraintes temporelles et de coordinations. Cependant, la construction du Template est progressive, ainsi ce dernier reste ouvert jusqu'à l'accomplissement de la dernière build activity dans le processus.

La figure suivante (partie 1 et partie 2) représente deux Instances Template pour l'exemple de workflow précédent.

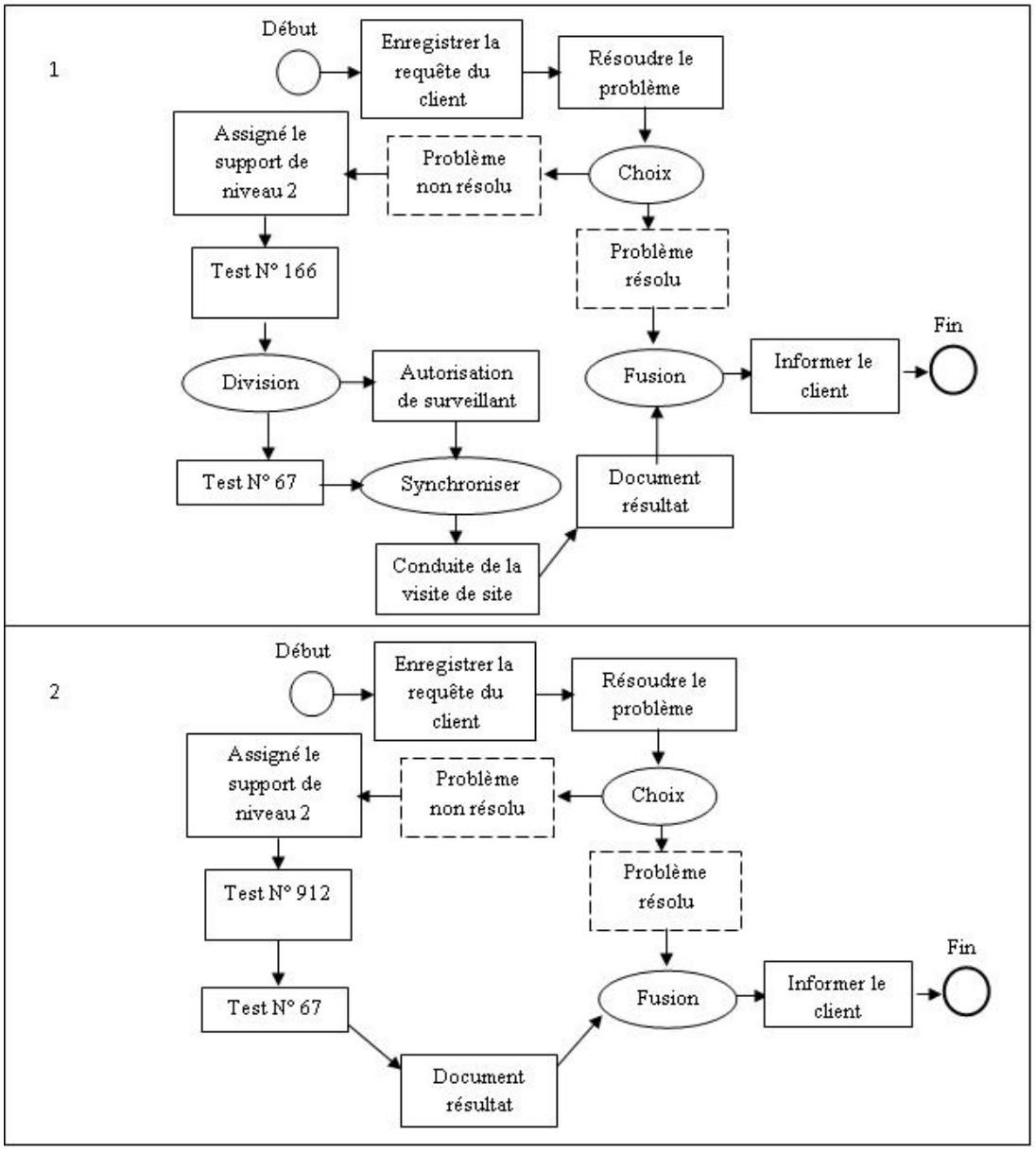



FIG. 3.3: Instances Template [37].

La force de cette approche se situe dans la séparation des spécifications de contraintes connues (qui doivent être imposées) et la spécification de contraintes difficiles à prédéterminer. Ce sont les utilisateurs qui ont cette connaissance. Cette séparation favorise la flexibilité dans les spécifications de processus métier (workflow).

## 3.7 Spécification de contraintes pour le workflow flexible

La spécification de contraintes dans un workflow flexible a pour but de caractériser la définition des poches de flexibilités au sein du modèle du workflow.

Dans [37], les auteurs ont défini deux classes de contraintes, nommées classe structurelle et classe retenue.

- **La classe structurelle :** Les contraintes définies dans cette classe imposent une restriction sur *comment les fragments seront composés dans le Template résultat*. On trouve dans cette classe les contraintes d'ordre (order), série (serial) et division (fork).
- **La classe retenue :** Les contraintes définies dans cette classe identifient les conditions dans lesquelles les fragments peuvent ou ne peuvent être présents dans le Template résultat. On trouve dans cette classe les contraintes d'inclusion et les contraintes d'exclusion.

Une méta-définition des contraintes de workflow donnée dans [37] est définie comme suit :

- Soit F un ensemble de fragments, et f ∈ F un fragment, ce dernier peut être une activité simple ou un sous processus.
- |F|= N, est le nombre de fragments dans F.
- $F_m \subseteq F$, avec m=1.2 ...N.
- C est un ensemble de contraintes
- Type-de-contrainte (ConstraintType) : C → $\Big\{$série, ordre, division, inclusion, exclusion$\Big\}$, $\forall c \in C$ le type-de-contrainte(c) représente le type de la contrainte c.
- W est le graphe de workflow, il représente de noyau du workflow (défini précédemment).
- P est la poche défini comme P=<W, F, C>.
- T est le Template résultat qui est prévu d'être dynamiquement construit à partir de la poche P. Dans le Template T les nœuds de coordination sont restreints aux constructions : $\Big\{$ *début, fin, synchroniseur , division* $\Big\}$.

Le résultat final de l'exécution d'une poche P est un Template T où une seule poche peut être transformée en plusieurs Templates (instances de workflow), et pour que ce dernier soit valide, tous les contraintes définies dans la poche P doivent être vérifiées dans T.

Une présentation détaillée de ces cinq types de contraintes est donnée dans ce qui suit :

### 3.7.1 Contrainte série

Une contrainte série est définie quand les fragments de workflow sont arrangés en série. Cependant le choix de l'ordre des fragments reste flexible et il est déterminé par l'utilisateur. Un exemple de composition valide en utilisant ce type de contrainte est représenté dans la figure suivante :

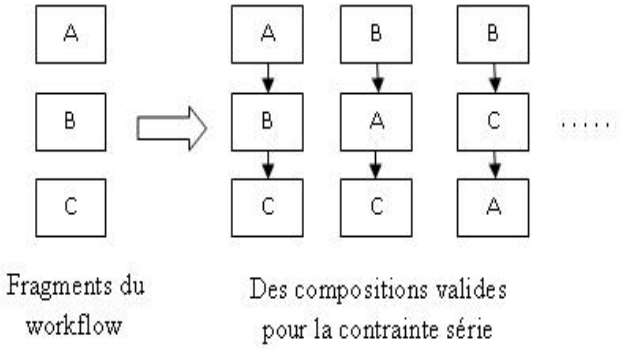



FIG. 3.4: Constructions série [37].

#### Notation

Soit $S(F_m)$ une contrainte série définit sur l'ensemble non vide de fragment $F_m$. $S(F_m)$ est représenté comment étant l'ensemble S($\left\{ f_1, f_2, \ldots, f_N \right\}$ ) ; $f_i \in F_m$ avec i=1..N.

#### Définition

Soit $S(F_m)$ une contrainte série et $F_k$ un sous ensemble de $F_m$ tel que tous les fragments dans $F_k$ sont présents dans le Template T, ce qui fais que $\forall f_i \in F_k$ , il existe un unique chemin entre eux dans T [37].

### 3.7.2 Contrainte d'ordre

La contrainte d'ordre est un cas particulier de la contrainte série, où les fragments doivent s'exécuter en série mais dans un ordre particulier.

Cependant, comme dans le cas des contraintes séries, les contraintes d'ordre n'imposent pas le placement consécutif de fragments (d'autres fragments peuvent exister).

**Notation**

Soit $O(F_m)$ une contrainte d'ordre, avec Fm un ensemble non vide de fragments dans F, $O(F_m)$ est représenté comme O($\left\{f_1, f_2, \ldots, f_N\right\}$). $f_i \in F_m$ avec i=1..N.

**Définition**

Soit $O(F_m)$ une contrainte d'ordre et $F_k$ un sous ensemble de $F_m$ tel que tous les fragments dans $F_k$ sont présents dans la Template T. $\forall f_i, f_{i+j} \in F_k$

avec i=1,2,. . .,.N-1, et j=1.2,. . .,N-i ; Il existe un chemin de $f_i$ à $f_{i+j}$ dans le Template T. Le choix des fragments est réalisé par l'utilisateur [37]. ,

### 3.7.3 Contrainte de division (fork)

La contrainte de division permet aux fragments choisis d'être exécuter en parallèle. La flexibilité de la définition est de pouvoir construire une division à partir de n'importe quel nombre de fragments [37] .

Soit un ensemble de fragments **F=(A ,B ,C ,D)**, et une contrainte **C** sur ces fragments , avec C= ***faire n'importe quel fragment de F mais pas plus que trois***. Ce comportement est représenté dans la figure suivante.

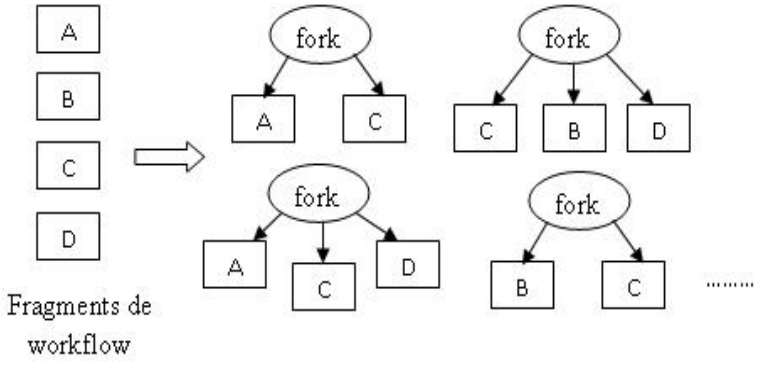



FIG. 3.5: Construction division (fork) [37] .

**Notation**

Soit $F(F_m)$ une contrainte de division (fork), avec $F_m$ un ensemble non vide de fragments dans F, $F(F_m)$ est représenté comme $\mathbf{F(}\{f_1, f_2, \ldots, f_N\}\mathbf{)}$. $f_i \in F_m$ avec i=1..N

**Définition**

Soit $F(F_m)$ une contrainte de division (fork) et $F_k$ un sous ensemble de $F_m$ tel que tous les fragments dans $F_k$ sont présents dans le Template T. $\forall f_i, f_j \in F_k$ , il existe pas de chemin entre eux dans le Template T. le choix des fragments est réalisé par l'utilisateur [37].

### 3.7.4 Contrainte d'inclusion

La contrainte d'inclusion identifie une dépendance entre deux ensembles de fragments. La présence ou l'absence des fragments dans un ensemble impose une restriction aux fragments dans le deuxième ensemble. La contrainte d'inclusion peut être complémentée par les contraintes série, ordre et division qui impose une restriction additionnelle sur la manière dont les fragments inclus doivent composés.

**Notation**

Soit $\mathbf{I(}F_p \mathbf{\ ,\ } F_m\mathbf{)}$ , avec $F_p$ et $F_m$ sont deux ensembles non vides de fragments de F. $\mathbf{I(}F_p \mathbf{\ ,\ } F_m\mathbf{)}$ est représenté par : I ($\left\{f_1, f_2, \ldots, f_Q\right\}$ , $\left\{f_1, f_2, \ldots f_n\right\}$), avec $f_i \in F_p$,i=1,2..Q. et $f_j \in F_m$ j=1,2 ...N.

**Définition**

Dans ce type de contrainte deux cas peuvent survenir :

- Si tous les fragments de $F_p$ sont présents dans le template T alors tous les fragments de $F_m$ doivent exister dans T.
- S'il existe des fragments de $F_p$ qui ne sont pas dans le Template T alors aucune règle n'est inférée sur $F_m$.

### 3.7.5 Contrainte d'exclusion

Les contraintes d'exclusion sont similaires aux contraintes d'inclusion, elles identifient une dépendance entre les fragments du workflow. Cependant ce type de contrainte interdit l'inclusion de fragments dans le Template résultat.

**Notation**

Soit $\mathbf{E}(F_p\ ,\ F_m)$ , avec $F_p$ et $F_m$ sont deux ensembles non vides de fragments de F. $\mathbf{E}(F_p\ ,\ F_m)$ est représenté par : E($\left\{f_1, f_2, \ldots, f_Q\right\}$,$\left\{f_1, f_2, \ldots, f_n\right\}$), avec $f_i \in F_p$, i=1,2 ...Q. et $f_j \in F_m$ j=1,2 ...N.

**Définition**

Comme avec la contrainte d'inclusion, deux cas peuvent survenir :

- Si tous les fragments de $F_p$ sont présents dans le Template T alors tous les fragments de Fm ne doivent pas exister dans T.
- S'il existe des fragments de $F_p$ qui ne sont pas dans le Template T alors aucune règle n'est inférée sur $F_m$.

## 3.8 Validation de contraintes

Après avoir défini les contraintes du modèle, l'aspect le plus important à présent c'est la validation. Puisque les contraintes servent pour contrôler la construction du Template, il est évident qu'elles ne doivent pas poser de conflits. En d'autres termes, pouvons-nous dire que les structures (instances) permises sous ces contraintes sont bien acceptables.

Une contrainte peut être invalide à cause de plusieurs facteurs comme le manque ou la perte de données, une violation temporelle et contradiction des règles métier (par exemple dans le cas de l'exécution de deux tâches dans le modèle dont la spécification de départ interdi l'existence de ces deux tâches dans une instance du processus).

Cependant, spécifier des contraintes incorrectes est équivalent à la construction d'un modèle de processus qui ne satisfait pas les contraintes de processus et par conséquent le but de ce dernier. Dans [37], une discussion détaillée des propriétés de ces contraintes est donnée ainsi qu'une démonstration de la validation de cet ensemble de contraintes.

## 3.9 Conclusion

Dans ce chapitre, nous avons abordé la notion de flexibilité dans le contexte de spécification de workflow, les poches de flexibilités comme moyen d'incorporer une certaine flexibilité dans les spécifications en fournissant plus de liberté à l'utilisateur dans sa spécification. Et enfin nous avons abordé le concept de spécification de contraintes pour le contrôle et la validation des définitions des poches dans un workflow flexible.

La recherche faite sur les poches de flexibilité permet d'avoir une idée pour la démarche à suivre afin d'incorporer un certain degré de contrôle dans les modèle flexibles qui souffrent de manque de contrôle ce qui entraîne des erreurs d'exécution.

Dans le chapitre suivant nous proposons un cadre méthodologique pour la modélisation hybride de workflow à l'aide du principe de poches de flexibilités, la théorie des graphes et la théorie d'Allen.

# Chapitre 4

# Approche hybride de modèlisation de workflow

## 4.1 Introduction

La spécification d'un procédé métier est une tâche très importante dans la procédure d'automatisation de processus d'entreprises. L'approche de modélisation déclarative malgré ses avantages dans la liberté qu'elle offre à l'utilisateur dans sa spécification et au système dans l'exécution, elle souffre de plusieurs erreurs lors de l'exécution et cela à cause de manque de contrôle.

Dans les chapitres précédents, nous avons abordé les deux méthodes de modélisation de workflow connues dans la littérature à savoir la méthode déclarative et la méthode impérative. Et nous avons étudié aussi le principe de poche de flexibilité qui est un moyen d'incorporer la flexibilité dans un modèle.

Dans ce présent chapitre, nous présentons notre approche hybride de modélisation de workflow en combinant entre les deux approches, impérative et déclarative. Elle consiste à partir d'un modèle déclaratif de départ d'effectuer des points de contrôle et de définir des poches impératives afin d'ajouter un certain degré de contrôle au modèle. Le processus est donné en détails dans les sections suivantes.

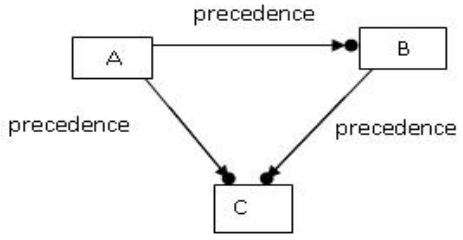
### 4.1.1 L'approche hybride de modélisation de workflow

Notre approche hybride permet d'avoir une méthode de modélisation composée. Elle combine les deux avantages : la flexibilité et le contrôle dans un seul formalisme.

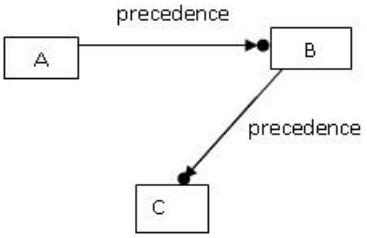
A partir d'un modèle déclaratif, on peut ajouter un certain degré de contrôle pour faire face aux problèmes posés précédemment, en effectuant **des points de contrôle** sur le modèle et en captant les **séquences impératives** dont les activités impliquées seront des activités incluses dans les poches que nous appelons **poches impératives**.

- **Les points de contrôle** : En utilisant les principes de la théorie des graphes et plus précisément avec un algorithme de détection de circuit adapté à notre cas qu'on a appelé **algorithme d'élimination de redondance** on peut éviter la possibilité d'erreur due à une redondance d'informations (ou la transitivité) dans la définition de **contraintes temporelles**. L'exemple suivant illustre ce comportement.

  **Exemple** : Soit un modèle déclaratif D=$\Big\{T, \theta\Big\}$ avec :
  T=$\Big\{A, B, C\Big\}$ et $\theta$ =$\Big\{precedence(A, B), precedence(B, C), precedence(A, C)\Big\}$.




  Lors de l'exécution, le système exécute la tâche A ensuite il peut déclencher l'exécution parallèle des deux tâches (B et C ) selon les deux contraintes (precedence(A,B) et precedence(A,C) ) et de ce fait violer la contrainte Precedence (B,C). Pour éviter ce type d'erreur un algorithme d'élimination de redondance est utilisé pour supprimer la contrainte Precedence(A,C) tout en gardant sa sémantique, et on aura comme résultat le schéma suivant :




- **Pour les poches impératives** : En utilisant la force de l'algorithme de chemin consistant (Path-Consistency) [27] et en définissant des relations d'intervalles selon ALLEN [9] [27] entre chaque paire de tâches du modèle déclaratif

on peut capter des séquences impératives séries ou parallèles et cerner les parties impératives pour mieux contrôler le modèle.

La figure suivante illustre tous le processus de notre approche.

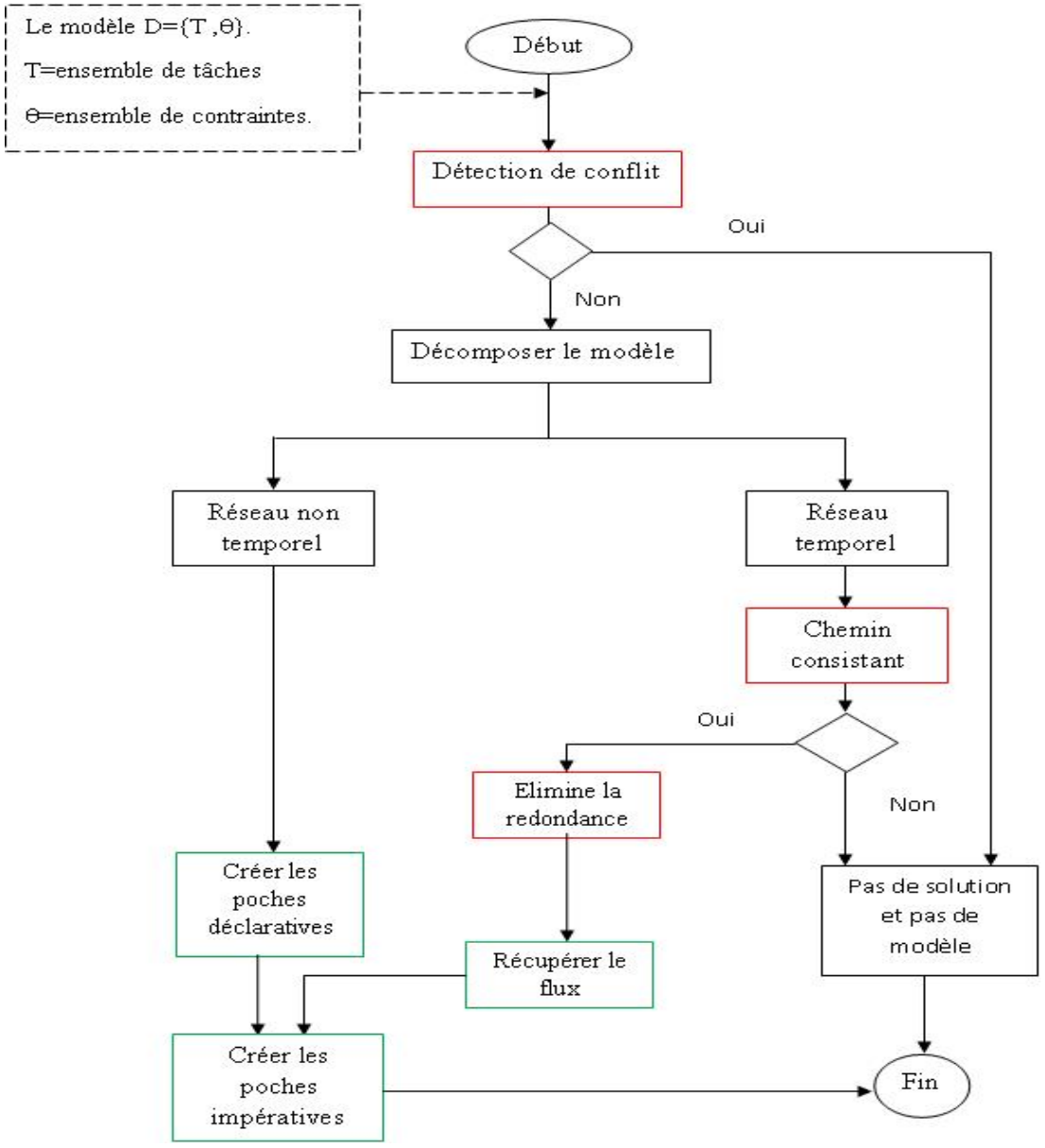



FIG. 4.1: Processus de modélisation hybride.

## 4.2 Détail de l'approche

L'approche est constituée de deux phases principales :

### 4.2.1 Phase 1 : La phase Préparatoire et exécution des points de contrôle

Dans cette phase, on procède à l'affectation des points de contrôle et la préparation de l'environnement pour le traitement qui sera effectué dans la deuxième phase. Cette phase permet de décomposer notre modèle afin de mieux le contrôler. Elle se compose de trois étapes.

#### 4.2.1.1 Première étape : Détection de conflit

Un conflit de contraintes peut exister dans la définition du modèle déclaratif s'il existe un circuit dans le graphe de contraintes ou s'il existe parmi l'ensemble de contraintes temporelles un conflit entre les contraintes série et parallèles.

**a. Cas d'un conflit de circuit** :

Dans ce cas, une erreur d'exécution est fortement probable si on a un circuit dans le graphe de contraintes du modèle.

**Exemple** : Soit la modèle D=(T,$\theta$), avec :

$$\text{T}=\Big\{A, B, C\Big\} \text{ et } \theta= \Big\{Precedence(A, B) \wedge Precedence(B, C) \wedge Precedence(C, A)\Big\}$$

L'exemple suivant illustre cette possibilité.

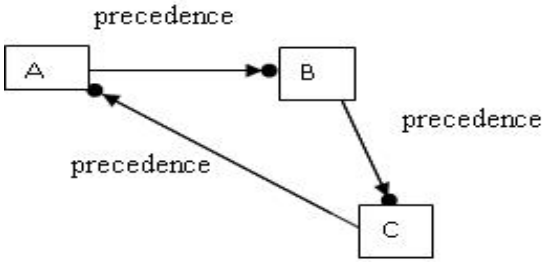



FIG. 4.2: Exemple d'un conflit de circuit.

**b. Cas d'un conflit entre les contraintes séries et parallèles** :

Ce type de conflit peut exister entre les contraintes temporelles. C'est à dire lorsqu'on a pour la même paire de tâches il existe une contrainte temporelle qui exprime le parallélisme et une autre contrainte série qui exprime un comportement série.

**Exemple** : Soit la modèle D=(T,$\theta$), avec :
T=$\Big\{A, B, C\Big\}$ et
$\theta = \Big\{Precedence(A, B) \wedge Precedence(B, C) \wedge Responded - Existence(B, A)\Big\}$.

Dans ce modèle, la contrainte Precedence (A,B) cause un conflit avec la contrainte Responded_Existence(B,A).

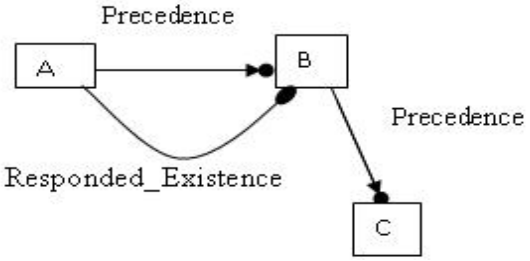



FIG. 4.3: Exemple d'un conflit de contraintes séries et parallèles.

#### 4.2.1.2 Deuxième étape : Décomposition de contraintes du modèle.

Le principe est de séparer les contraintes du modèle déclaratif en contraintes temporelles (dont on peut avoir un raisonnement temporel),et non-temporelles.

Le tableau suivant représente les formules de relations et de négations de DecSerFlow ( [40]) séparées en deux types, temporelles et non temporelles.

| **Contraintes temporelles** | **Contraintes non temporelles** |
|---|---|
| Responded_Existence | Responded_Absence |
| Co_existence | Not_Co_Existence |
| Response | Negation_Reponse |
| Precedence | Negation_Precedence |
| Succession | Negation_succesion |
| Alternate_response | Negation_Alternate_Response |
| Alternat_precedence | Negation_Alternate_Precedence |
| Alternate_succession | Negation_Alternate_succession |
| Chain_Response | Negation_Chain_Response |
| Chaine_precedence | Negation_Chain_Precedence |

TAB. 4.1: Les deux types de contraintes temporelles et non temporelles.

Soit D le modèle déclaratif de départ avec D=(T, $\Theta$). La décomposition de ce modèle, nous donne les deux sous modèles suivants : Tem et Non_Tem.

Avec $Tem = (T_1, \Theta_1) et Non\_Tem = (T_2, \Theta_2)$.

La figure suivante représente respectivement le réseau de contraintes global R, le sous réseau R1 qui englobe les contraintes de type temporel et le sous réseau R2 qui englobe les contraintes de type non temporel.

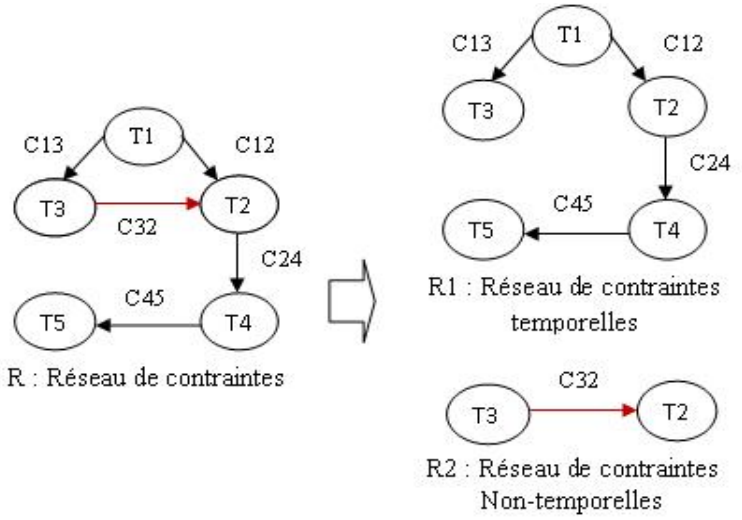



FIG. 4.4: Exemple d'une décomposition d'un réseau de contraintes R.

Pour cet exemple D=($\{T1, T2, T3, T4, T5\}, \{C_{12}, C_{13}, C_{32}, C_{24}, C_{45}\}$). La décomposition en R1 et R2 nous donne les deux sous modèles Tem et Non_Tem avec :
Tem=($\{T1, T2, T3, T4, T5\}, \{C_{12}, C_{13}, C_{24}, C_{45}\}$).
Non_Tem=($\{T2, T3\}, \{C_{32}\}$).

Pour mieux comprendre le principe nous donnons dans ce qui suit un exemple concret d'un processus métier modéliser avec ConDec [33].

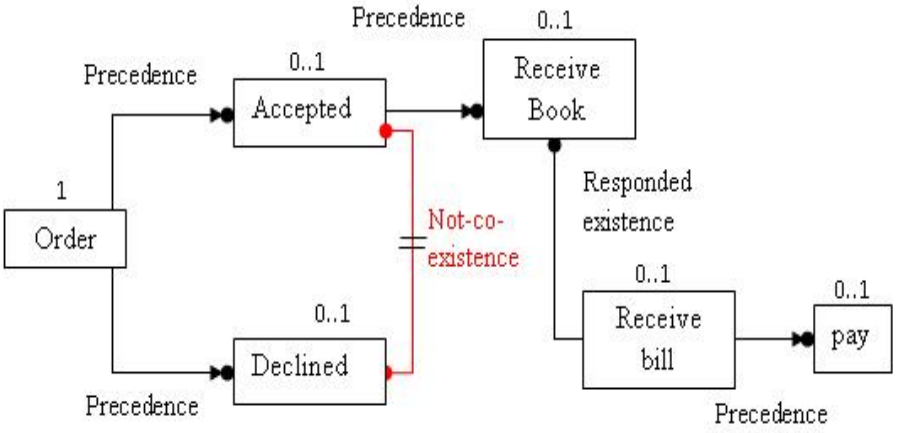
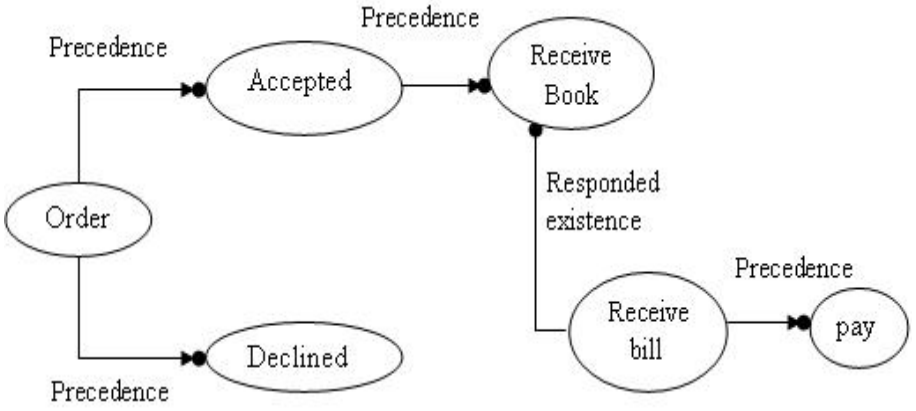
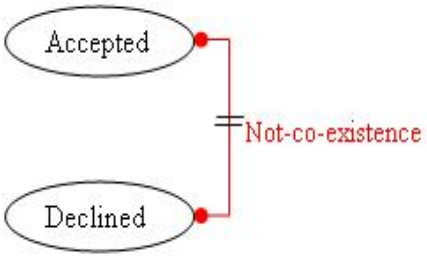



FIG. 4.5: Processus d'achat d'un livre avec ConDec [33].

On peut décomposer ce modèle en deux réseaux suivants :




**R1**: Réseau de contraintes temporelles




**R2**: Réseau de contraintes non temporelles

FIG. 4.6: Les réseaux de contraintes temporelles et non temporelles résultat de la décomposition

### 4.2.2 Phase 2 : La phase de Traitement

Dans cette phase, on va effectuer les transformations de base pour notre modèle déclaratif de départ afin de capter le comportement impératif dans les poches impératives.

#### 4.2.2.1 Première étape : Raisonner sur le réseau de contraintes temporelles

**A. Le réseau de contraintes temporelles : *Représentation des contraintes temporelles par les relations d'intervalles d'Allen.***

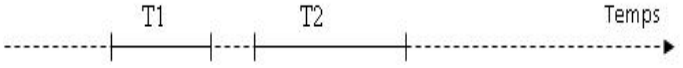
Afin de raisonner sur le réseaux de contraintes temporelles et de pouvoir utiliser l'algorithme de chemin consistant, il faux définir pour chaque paire de tâches un intervalle correspondant selon la définition d'Allen [9].

En ce basant sur les contraintes Template définit dans le modèle déclaratif graphique DecSerFlow [40] (ConDec [33]) et les relations d'intervalle d'ALLEN [9], on traduit chaque formule de relation définit pour chaque paire de tâches en une relation d'intervalle et on définit un réseau de contraintes temporelles pour l'ensemble de ces tâches.

**• Traduire les relations temporelles en relation d'intervalle d'Allen**

Dans le cas d'un processus métier, les tâches peuvent s'exécuter en série ou en parallèle :

- **L'exécution série** : Deux tâches T1 et T2 s'exécutent en série si l'une est exécutée avant l'autre.




Exécution serie de deux tâches

On peut représenter ce comportement série entre deux tâches avec les relations d'intervalles d'Allen avec l'ensemble $\{b, bi, m, mi\}$ ou $\{before, meet, after, met-by\}$. Le comportement série du schéma précédent est représenté avec la relation d'intervalle T1 $\{b\}$ T2.

- **L'exécution parallèle** : Deux tâches T1 et T2 peuvent s'exécuter en parallèle si elles commencent ou finissent en même temps ou leur exécutions

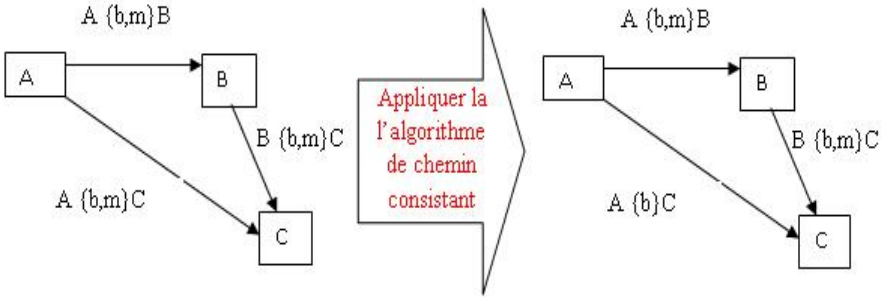
s'intercalent dans le temps. Le comportement parallèle est représenté avec les relations d'intervalles d'Allen de l'ensemble $\{s, d, f, eq, o, si, di, fi, oi\}$ ou $\left\{\begin{array}{c} starts, during, finishes, equal, overlaps, \\ started-by, contains, finished-by, overlapped-by \end{array}\right\}$.

Nous donnons dans l'annexe D un tableau représentant pour chaque formule de relation sa relation d'intervalle équivalente.

**• Appliquer l'algorithme de chemin consistant pour les contraintes temporelles :**

Après la traduction de chaque formule de relation du modèle déclaratif en une relation d'intervalle d'Allen, et la représentation de ces dernière sous forme d'un réseau de contraintes temporelles correspondant (réseau de relation d'intervalle d'Allen), l'application de l'algorithme de chemin consistant [27] présenté dans le chapitre 2 aura lieu afin de filtrer les domaines de chaque variable (tâche) et définir si le CSP temporelle est consistant.

**Exemple** : L'algorithme de chemin consistant permet de filtrer les relations pour chaque paire de tâches d'un modèle. Voici un exemple explicatif.




Exemple d'application de l'algorithme de chemin consistant

**• Appliquer l'algorithme d'élimination de redondance dans le réseau de contraintes temporelles comme étant un point de contrôle :**

Afin d'éviter quelques erreurs d'exécution causées par la redondance d'informations, l'application de l'algorithme d'élimination de redondance permet de résoudre ce problème.

**Algorithm 4.1** Ilimination-de-redondance

1: **Soit R1 le sous réseau temporel du réseau globale R**
2: **F1= (x,y), x,y ∈ R1, et F1 la relation de flux dans le réseau R1**
3: **Début**
4: $\forall(x, y) \in F1$
5: Visiter(x,y)
6: **Fin.**

La procédure **visiter** permet de supprimer les arcs qui causent une redondance et ainsi une erreur d'exécution.

**Procedure. 4.2** Visiter (x,y)

1: **Début**
2: **si** (Non marque(x,y)) **alors**
3: marque (x,y)← vrai
4: **si** $\exists z \in R1 \wedge (y, z) \in F1$ **alors**
5: **si** Non marque(y,z) **alors**
6: marque(y,z)← vrai
7: **si** (x,z) ∈ F1 **alors**
8: supprimer(x,z)
9: **sinon**
10: y ← z
11: **si** ∃ arc non-marque **alors**
12: aller à 4.
13: **finsi**
14: **finsi**
15: **finsi**
16: **finsi**
17: **finsi**
18: **Fin.**

#### 4.2.2.2 Deuxième étape : Construire des poches impératives

Dans cette étape, on procède à la création des poches déclaratives et impératives à partir du modèle déclaratif de départ.

**a. Encapsuler les parties déclaratives non temporelles :**
Notre approche pour cette étape consiste à encapsuler les contraintes déclaratives non temporelles du modèle de départ dans des poches déclaratives afin de ne pas affecter l'exécution des autres tâches et causer une erreur d'exécution. Et c'est à l'utilisateur de choisir pour chaque poche déclarative la (les) tâche(s) à exécuter. La figure suivante représente deux poches déclaratives dont chacune contient une tâche

déléguée. Cette dernière sert à déléguer les tâches participantes à la définition de ces poches.

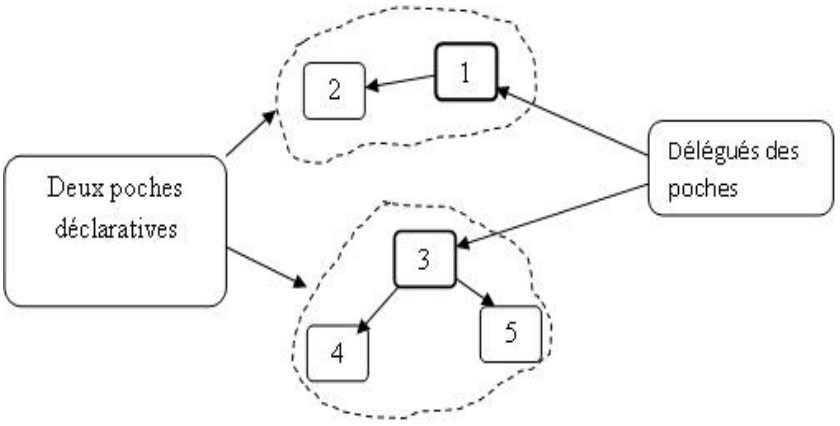



FIG. 4.7: Poches déclaratives pour le réseau non-temporel.

Un algorithme de construction de ces poches déclaratives est donné comme suit.

**Algorithm 4.3** Poches-déclaratives

1: **R2 : Le sous réseau de contraintes non-temporelle**
2: **L** $= \cup C_{ij}, C_{ij} \in R2$ Contrainte-non-temporelle.
3: **Exist[i]=faux, i=1..n,** n est le nombre de tâches du modèle D.
4: **poche-d[i]=Faux, i=1..n**. Construire une poche declarative dont la tâche principale est i.
5: **participe-d[i]=**$\emptyset$. Précise dans quelle poche déclarative participe une tâche.
6: **Début**
7: $\forall C_{ij} \in L$
8: **si** (Not Existe[i]) **alors**
9: Existe[i]← vrai
10: **si** (Not Existe[j]) **alors**
11: Existe[j]← vrai
12: poche-d[i]← i
13: participe-d[i]← i
14: participe-d[j]← i
15: **sinon**
16: participe-d[i]← participe-d[j]
17: **finsi**
18: **finsi**
19: **Fin.**

**b. Récupérer le flux pour chaque tâche du modèle D de départ (prédécesseurs, successeurs, parallèles.)**

A partir du réseau temporel, on peut récupérer pour chaque tâche ses tâches successeurs, prédécesseurs, et parallèles. La récupération de dépendances de flux entre les tâches nous permet de créer les poches impératives dans le point suivant.

**c. Construire les poches impératives : Construire les poches impératives et le modèle hybride final.**
Après la construction des poches déclaratives et la récupération des dépendances de flux entre les tâches dans les étapes précédentes, on procède à la construction du modèle hybride.

La construction des poches impératives consiste pour chaque tâche (source, début de processus) dont il n'existe pas de prédécesseurs à vérifier ses tâches successeurs et ses tâches parallèles qui ne participent pas à la définition d'une poche déclarative et ainsi construire la poche impérative avec ces tâches et dont la tâche source sera la tâche déléguée de la poche.

La définition des poches impératives dans un modèle conçu au départ comme déclaratif, permet de cerner et de fermer le comportement impératif afin de mieux contrôler le modèle, guider le système dans sa tâche d'exécution, et par conséquent la possibilité d'erreurs diminue.

Le processus de création de poches impératives est donné par l'algorithme suivant :

**Algorithm 4.4** Poches-Impératives

```
1: Exist-D[i]=Faux, i=1..n, indique si une tâche participe dans une poche déclarative.
2: Exist-I[i]=Faux, i=1..n, indique si une tâche participe dans une poche Impérative.
3: poche-I[i]=Faux, i=1..n, construire une poche Impérative dont la tâche principale est i.
4: pred[i]=succ[i]=parall[i]=∅, i=1..n, prédécesseurs, successeurs,parallèles d'une tâche i initilisés à vide.
5: participe-I[i]=∅, précise dans quelle poche Impérative participe une tâche.
6: Début
7: si Chemin-consistant-R1 alors
8:    Poches-déclaratives().
9:    Ilimination-de-redondance().
10:   récupérer-le-flux().
11:   pour k = 1, ..., n faire
12:      si (pred[i] = ∅) alors
13:         si (Not Existe-D[i]) alors
14:            si (Not Existe-I[i]) alors
15:               Existe-I[i]← vraie.
16:               poche-I[i]← vraie.
17:               tantque (succ[i]≠ ∅) faire
18:                  j ← succ[i].
19:                  succ[i]← succ[i]-1.
20:                  si (Not Existe-D[j]) alors
21:                     si (Not Existe-I[j]) alors
22:                        Existe-I[j]← vraie.
23:                        participe-I[j]← i.
24:                     finsi
25:                  finsi
26:               fin tantque
27:               tantque (parall[i]≠ ∅) faire
28:                  j ← parall[i].
29:                  parall[i]← parall[i]-1.
30:                  si (Not Existe-D[j]) alors
31:                     si (Not Existe-I[j]) alors
32:                        Existe-I[j]← vraie.
33:                        participe-I[j]← i.
34:                     finsi
35:                  finsi
36:               fin tantque
37:            finsi
38:         finsi
39:      finsi
40:   fin pour
41: finsi
42: Fin.
```

## 4.3 Conclusion

Nous avons présenté dans ce chapitre les différentes étapes de notre approche de modélisation hybride de workflow. Dans le but d'avoir un équilibre entre la flexibilité des modèles déclaratifs et le contrôle offert par les modèles impératifs on est parti du principe d'avoir un modèle déclaratif flexible et de lui ajouter un contrôle en effectuant des transformations et des traitements sur ce modèle.

Pour cela, deux phases ont été définit qui permettent d'effectuer des points de contrôle sur le modèle flexible pour détecter l'existence d'un conflit dans la définition de contraintes et d'encapsuler d'un côté les parties déclaratives dont la connaissance de leur informations est connue seulement par l'utilisateur et dont nous donnons par notre approche la possibilité à l'utilisateur de choisir ses tâches à l'aide du principe de poches déclaratives, et d'un autre côté les parties impératives dans des poches impératives afin d'encapsuler et de fermer le comportement impératif pour mieux contrôler le modèle.

Le chapitre suivant expose la mise en œuvre du cadre méthodologique proposé en vue démontrer la faisabilité de l'approche sur une étude de cas d'un processus métier proposé par Pesic et Aalst.

# Chapitre 5

# Implémentatin de l'approche hybride de modèlisation de workflow

## 5.1 Introduction

NOUS avons présenté dans le chapitre précédent les étapes de notre approche hybride. Dans ce qui suit, on va décrire l'implantation de l'approche par une étude de cas d'un exemple d'un processus métier d'achat de livres proposé par Pesic et Aalst [33].

## 5.2 Implémentation et mise en oeuvre par une étude d'un cas

Avant de procéder à la première phase, on doit introduire le modèle déclaratif. A l'aide d'une interface graphique, l'utilisateur peut introduire le modèle en donnant l'ensemble de ses tâches et de ses contraintes sous forme de formules de relations (contraintes temporelles) et de négations (contraintes non temporelles).

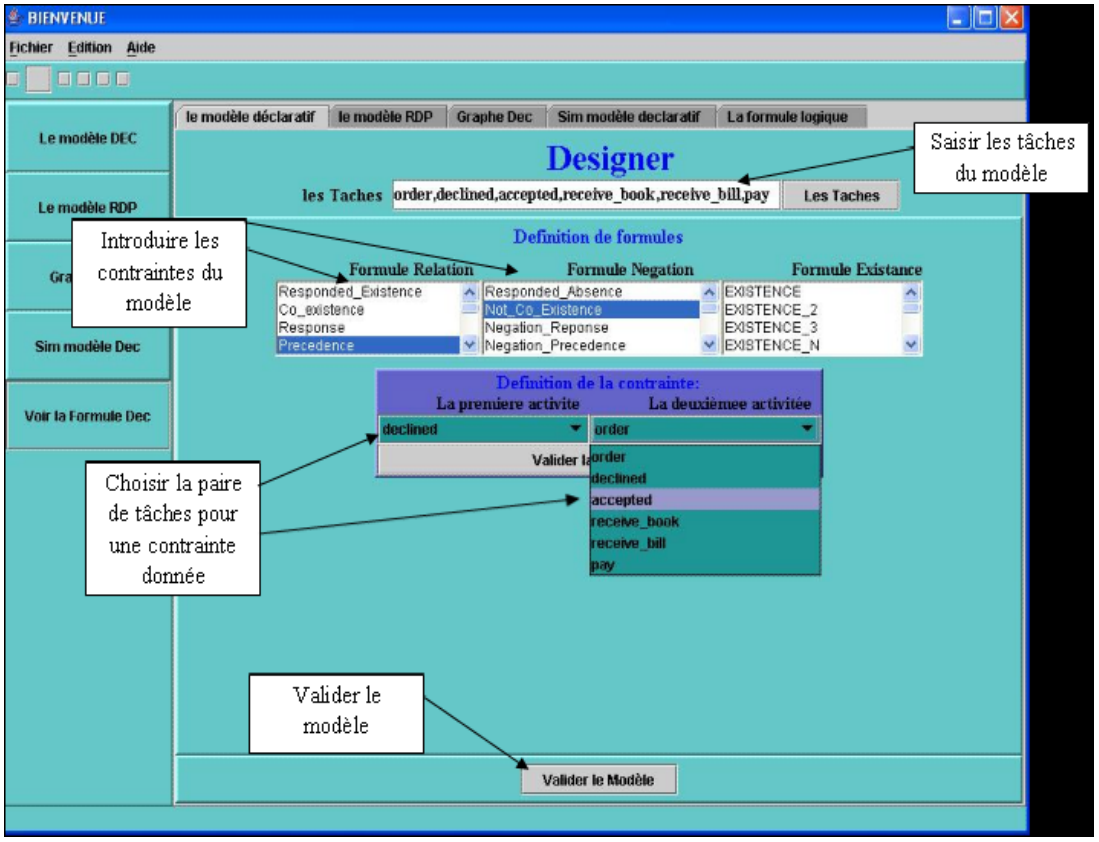



FIG. 5.1: Interface de spécification du modèle Déclaratif de départ.

A partir de cette interface, on aura le modèle déclaratif **D** suivant :

D= (T,Θ) avec : T est l'ensemble de tâches (activités), et Θ est l'ensemble de contraintes du modèle.

$$\mathrm{T}= \Big\{Order, Declined, Accepted, Receive\_book, Receive\_bill, Pay\Big\}.$$

$$\Theta = \left\{ \begin{array}{c} Precedence(Order, Declined), Precedence(Order, Accepted), \\ Not\_co\_existence(Declined, Accepted), Precedence(Accepted, Receive\_book), \\ Respended\_existence(Receive\_book, Receve\_bill), \\ Precedenc(Receive\_bill, Pay) \end{array} \right\}$$

### 5.2.1 Phase 1 : La phase Préparatoire et exécution des points de contrôle

Après l'introduction du modèle déclaratif, on procède aux différentes étapes de notre approche en commençant par les points de contrôles.

#### 5.2.1.1 Première étape : Détection de conflit

Comme déjà expliqué, des points de contrôles sont effectués sur le modèle afin de détecter l'existence d'une erreur dans la spécification de contraintes. Un algorithme de détection de circuit est implémenté [12] pour détecter l'existence de conflit cas d'un circuit.

Pour le conflit *cas de contraintes séries et parallèles*, une simple vérification de contraintes par un algorithme de vérification peut être effectué afin d'éviter ce type d'erreurs.
Pour l'exemple de Pesic et Aalst [33], le modèle ne comporte pas de conflits.

#### 5.2.1.2 Deuxième étape : Décomposition des contraintes du modèle.

Dans cette étape on décompose les contraintes du modèle en contraintes temporelles et non temporelles afin d'effectuer les traitements sur les deux types de réseaux résultant.

Pour l'exemple de Pesic et Aalst, on aura la décomposition suivant :

- R1=Tem=$(T_1, \Theta_1)$ avec :
  $T_1 = \Big\{Order, Accepted, Declined, Receive\_book, Receive\_bill, Pay\Big\}$.

$$\Theta_1=\left\{\begin{array}{c} Precedence(Order, Declined), Precedence(Order, Accepted), \\ Precedence(Accepted, Receive\_book), \\ Responded\_existence(Receive\_book, Receive\_bill), \\ Precedence(Receive\_bill, Pay) \end{array}\right\}.$$

- R2=Non_Tem=$(T_2, \Theta_2)$ avec :

  $T_2 = \Big\{Accepted, Declined\Big\}$.

  $\Theta_2=\Big\{Not\_co\_existence(Accepted, Declined)\Big\}$.

### 5.2.2 Phase 2 : La phase de Traitement.

#### 5.2.2.1 Première étape : Raisonner sur le réseau de contraintes temporelles.

- **Représentation des contraintes temporelles par les relations d'intervalles d'Allen :**

Voici le réseau temporel R1 avec les relations d'intervalles d'Allen.

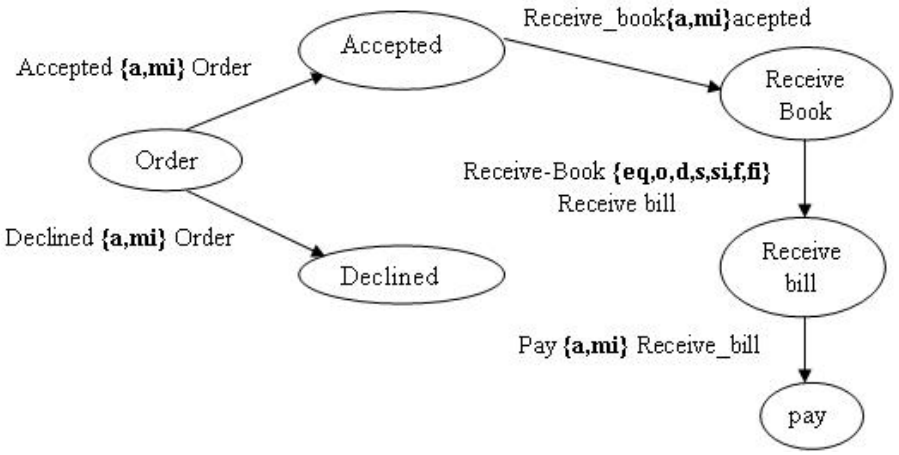



FIG. 5.2: Un réseau de contraintes temporelles avec les intervalles d'Allen

- **Appliquer l'algorithme de chemin consistant pour les contraintes temporelles :** L'application de l'algorithme de chemin consistant permet de filtrer les relations de chaque paire de tâches. Dans l'exemple de la figure précédente, les relations d'intervalles reste les mêmes.
- **Appliquer l'algorithme d'élimination de redondance dans le réseau de contraintes temporelles :** L'algorithme d'élimination de redondance est effectué dans le cas de la présence d'une transitivité, ce qui n'est pas le cas de notre exemple.

#### 5.2.2.2 Deuxième étape : Construire des poches impératives.

- **Encapsuler les parties déclaratives non temporelles :** Dans cette étape, on procède à la création des poches déclaratives afin d'encapsuler les parties déclaratives et de les cerner. Pour notre modèle, on a une seul poche déclarative : Non_Tem= $(T_d, \Theta_d)$ avec :

  $T_d = \Big\{Accepted, Declined\Big\}$.
  $\Theta_d = \Big\{Not\_co\_existence(Accepted, Declined)\Big\}$.
  Dans cette poche lors de l'exécution, l'utilisateur aura le choix entre les deux activités Accepted ou Declined.

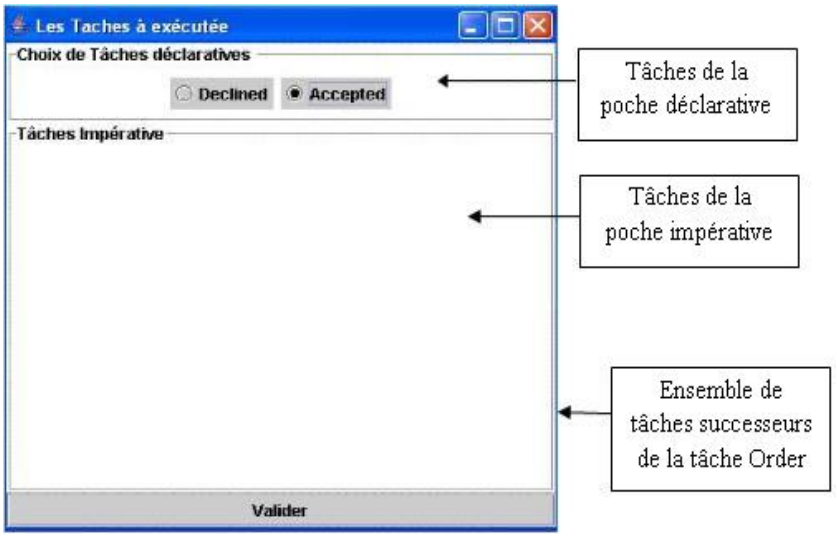



FIG. 5.3: Interface pour le choix de la tâche à exécuter.

- **Construire les poches impératives : construire les poches impératives et le modèle hybride final :**
  La construction des poches impératives nécessite la récupération de flux. Pour chaque tâche on récupère ses tâches successeurs, parallèles et prédécesseurs.

| **Tâche** | **Prédécesseurs** | **Successeurs** | **Parallèles** |
|---|---|---|---|
| Order | \ | Accepted, Declined | \ |
| Accepted | Order | Receive_book | \ |
| Declined | Order | \ | \ |
| Receive_book | Accepted | \ | Receive_bill |
| Receive_bill | \ | Pay | Receive_book |
| Pay | Receive_bill | \ | \ |

TAB. 5.1: Tableau de flux pour le modèle déclaratif de Pesic et Aalst.

Voici un schéma représentant les deux types de poches (impérative et déclarative) créées pour notre modèle déclaratif de départ D.

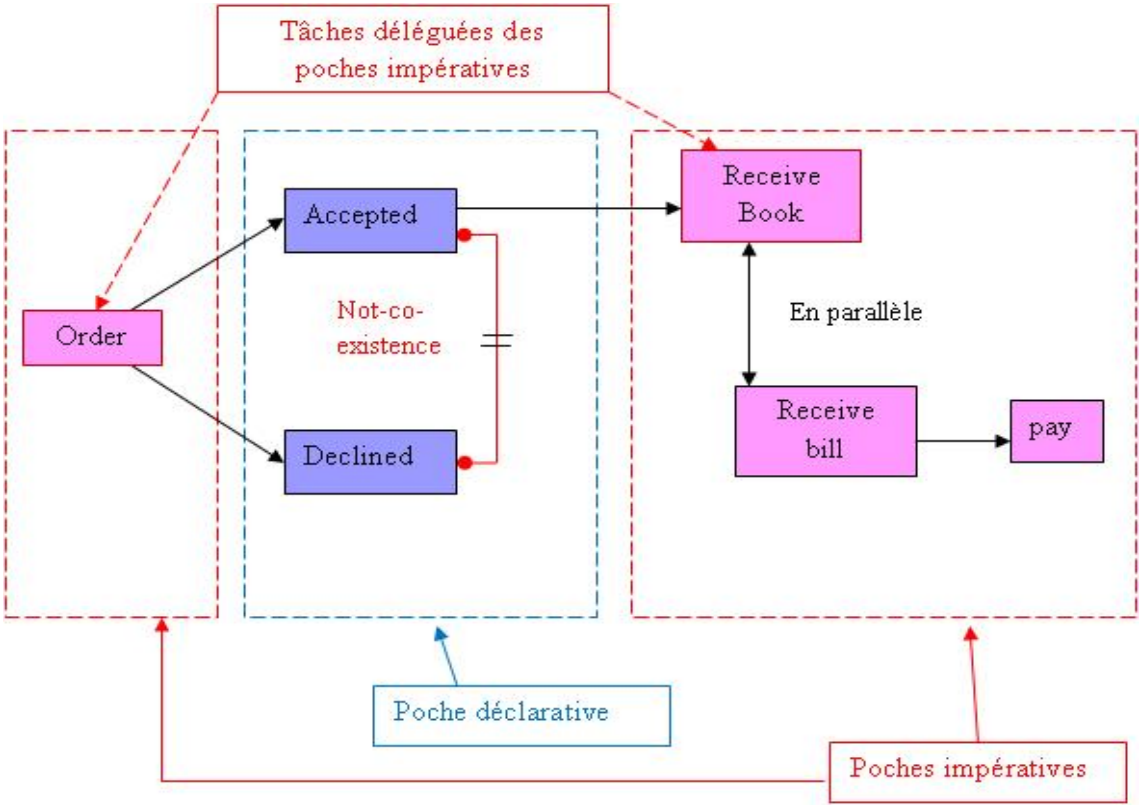



FIG. 5.4: Un schéma représentant les poches impératives et déclaratives du modèle D.

A l'aide d'un algorithme d'exécution du modèle, on aura les deux instances d'exécution représentées dans les schémas suivants selon le choix effectué par l'utilisateur.

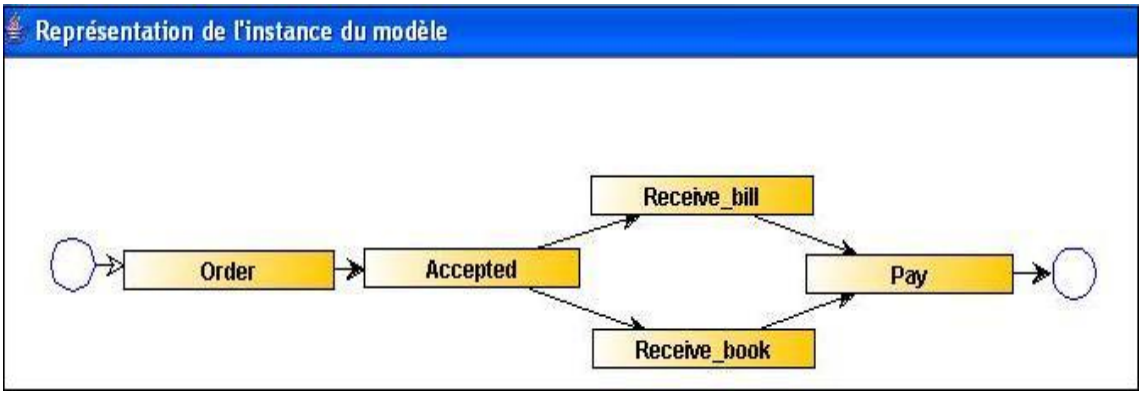



FIG. 5.5: Une exécution dans le cas d'un choix de la tâche Accepted.

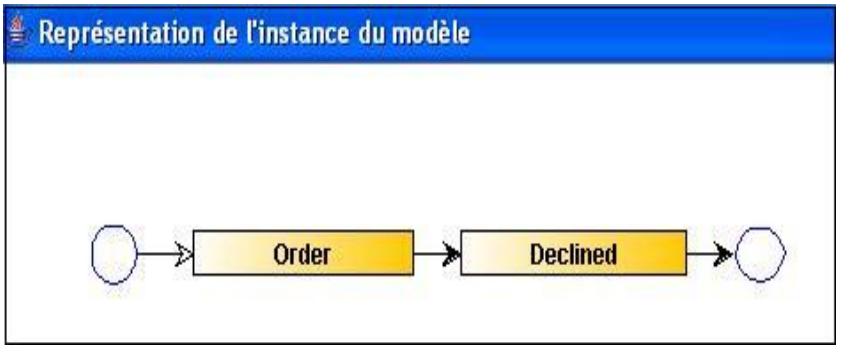



FIG. 5.6: Une exécution dans le cas d'un choix de la tâche Declined .

Selon le choix de l'utilisateur, voici deux fichiers dont on a sauvegardé les tâches exécutées avec leurs temps d'exécution, leur date de début et de fin.

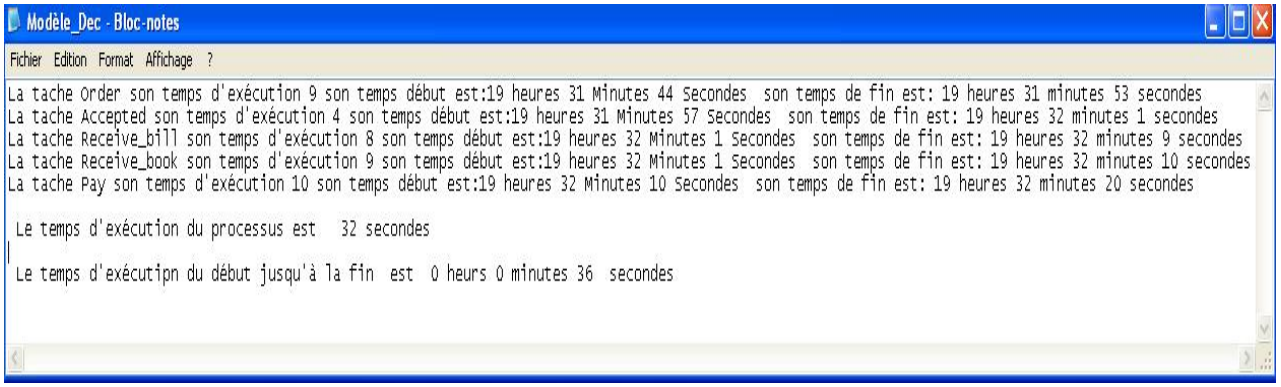



FIG. 5.7: Résultats de l'exécution du modèle D avec le Choix de la tâche Accepted .

Dans ce fichier la différence entre le temps d'exécution 32 et 36 est le temps d'attente de l'utilisateur pour faire son choix de la tâche Accepted.

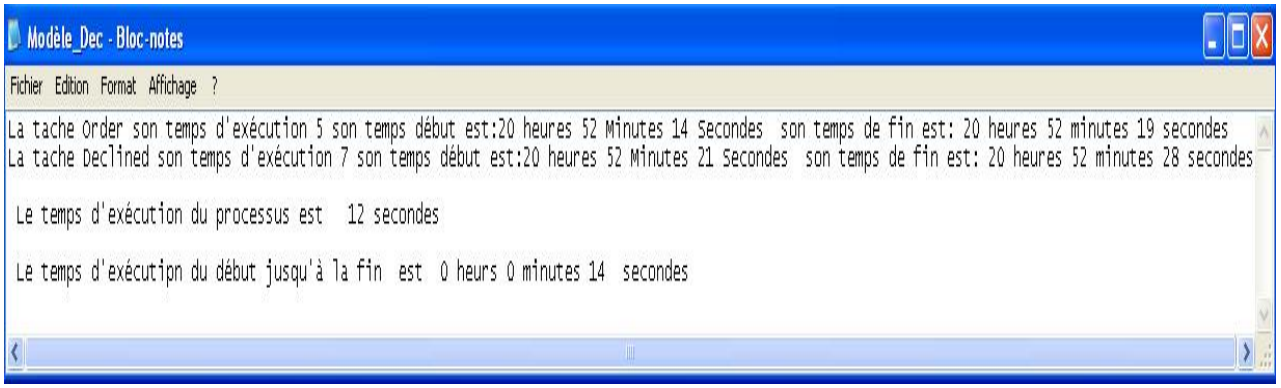



FIG. 5.8: Résultats de l'exécution du modèle D avec le Choix de la tâche Declined .

La même chose que dans le fichier de la figure 5.6, la différence entre le temps d'exécution 12 et 14 est le temps d'attente de l'utilisateur pour le choix de la tâche Déclined.

## 5.3 Exemple comparatif

Pour montrer l'efficacité de notre technique, nous avons choisi de la comparer avec une méthode impérative de modélisation qui est la méthode par réseau de pétri. Pour cela nous avons utilisé le même exemple de Pesic et Aalst.

La comparaison se base sur le temps d'exécution et la possibilité du choix ou la flexibilité offerte à l'utilisateur.
Voici le schéma du modèle de réseau de pétri proposer par Pesic et Aalst.

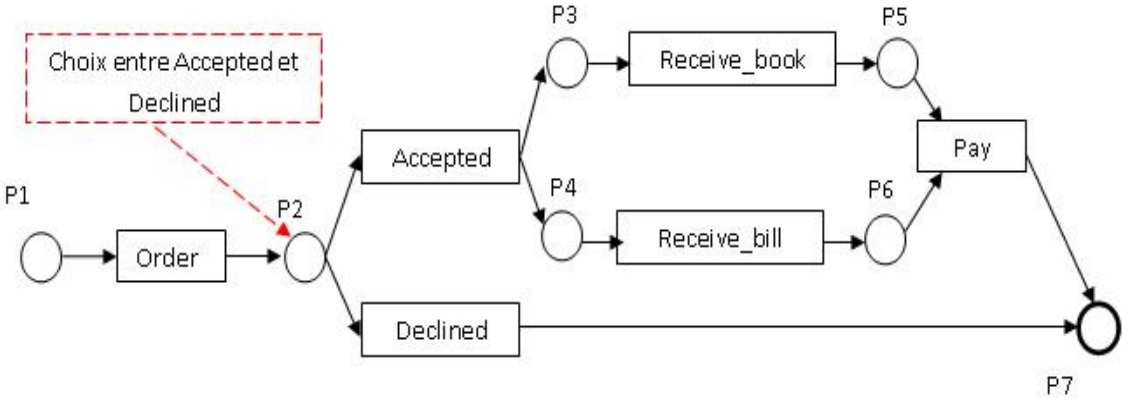



FIG. 5.9: Modèle de réseau de pétri pour l'achat de livre [33].

Dans la place p2, le système exécute une verification d'une donnée ou un calcul pour spécifier le choix du chemin à suivre. Dans notre cas on a utilisé une fonction pour générer le choix du chemin automatiquement.

## Introduire le modèle de RDP

Pour la modélisation par un réseau de pétri de notre processus, nous avons besoin des deux matrices d'incidences. Une matrices d'incidences avant $W^-$ (Places-Transitions) et une matrices d'incidences arrière $W^+$(Transition-Places) pour pouvoir introduire les dépendances de flux entre les tâches.

| **La matrice d'incidence avant $W^-$** | | | | | | |
|---|---|---|---|---|---|---|
| P \T | Order | Declined | Accepted | Receive_book | Receive_bill | Pay |
| P1 | 1 | 0 | 0 | 0 | 0 | 0 |
| P2 | 0 | 1 | 1 | 0 | 0 | 0 |
| P3 | 0 | 0 | 0 | 1 | 0 | 0 |
| P4 | 0 | 0 | 0 | 0 | 1 | 0 |
| P5 | 0 | 0 | 0 | 0 | 0 | 1 |
| P6 | 0 | 0 | 0 | 0 | 0 | 1 |
| P7 | 0 | 0 | 0 | 0 | 0 | 0 |

Tab. 5.2: La matrice d'incidence avant $W^-$.

| **La matrice d'incidence arrière $W^+$** | | | | | | | |
|---|---|---|---|---|---|---|---|
| T \P | P1 | P2 | P3 | P4 | P5 | P6 | P7 |
| Order | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| Declined | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Accepted | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| Receive_book | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| Receive_bill | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| Pay | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

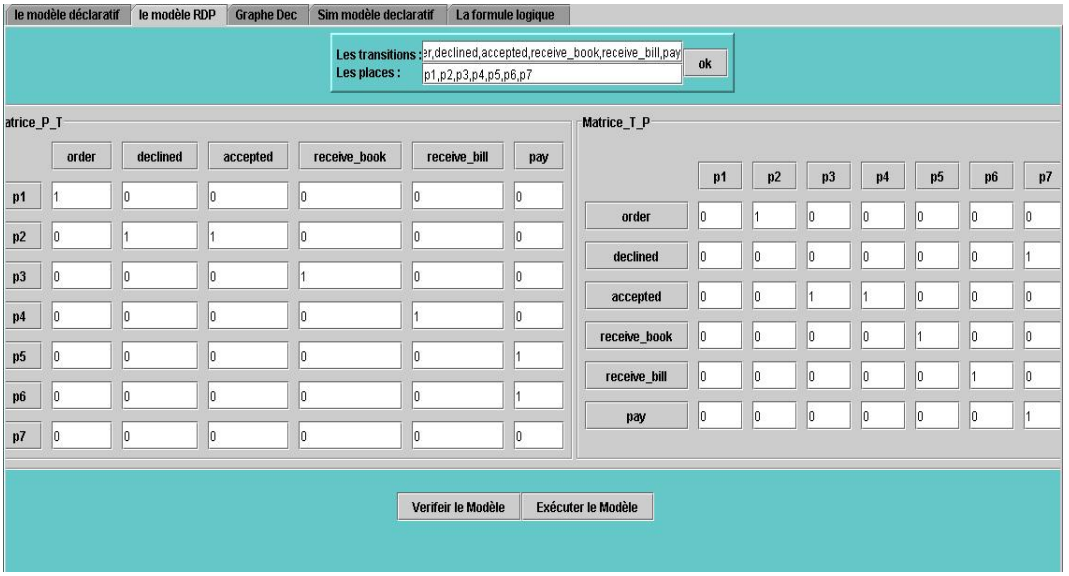
Tab. 5.3: La matrice d'incidence arrière $W^+$.

## Verification du modèle

Avant l'exécution du modèle on doit vérifier quelques propriétés comme la présence du blocage et l'existence de boucle qui pourront causer un blocage du système lors de l'exécution. Un algorithme de calcul de graphe de marquage [15] est implémenté permettant de vérifier ces propriétés en calculant le graphe de marquage du

réseau de pétri.

L'interface suivante permet à l'utilisateur d'introduire les deux matrices d'incidences et de verifier le modèle.




Interfece de modèlisation par RDP

## Exécution du modèle.

Après la vérification, on procède à l'exécution du modèle à l'aide d'un algorithme d'exécution et on aura comme résultat les deux fichiers suivants selon le choix effectuer par le systéme pour les deux tâches Accepted et Declined. Dans ces fichiers on a enregistré les états du graphe de marquage ainsi que le temps d'exécution de chaque tâche, son temps de début et de fin d'exécution.

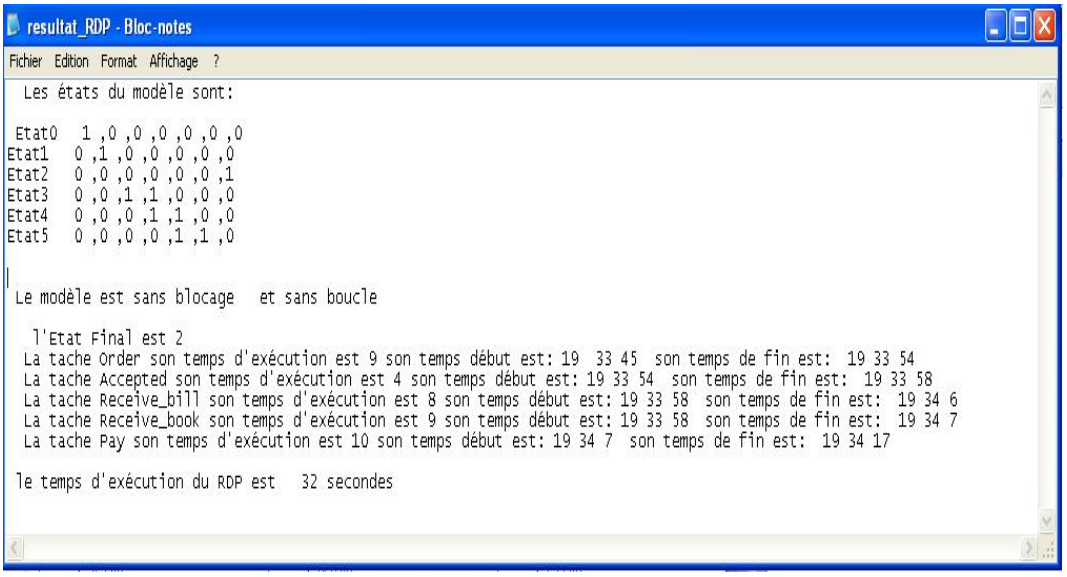



FIG. 5.10: Résultats d'exécution du modèle de réseau de pétri pour le choix de la tâche Accepted.

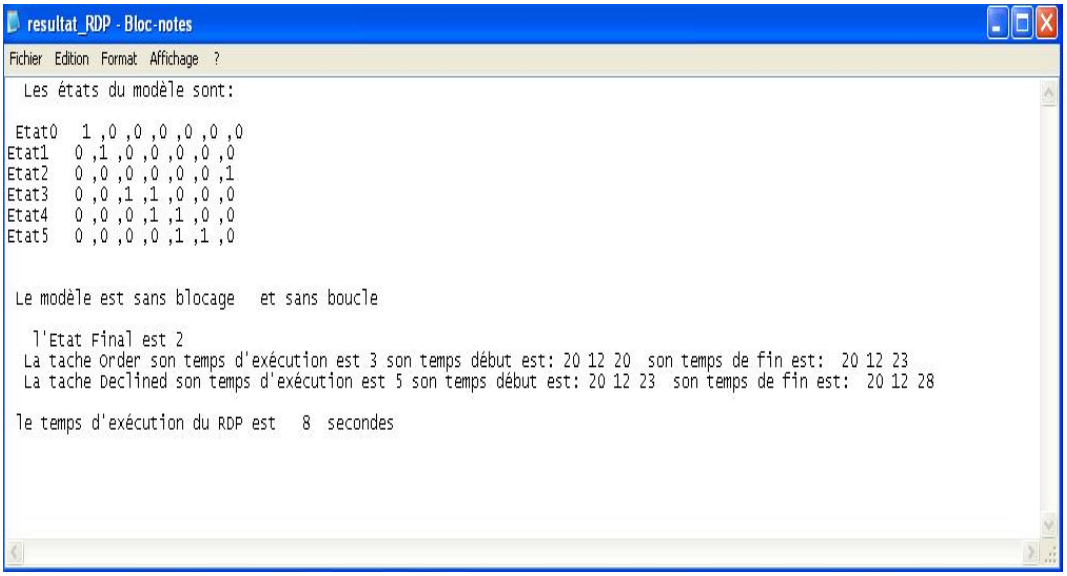



FIG. 5.11: Résultats d'exécution du modèle de réseau de pétri pour le choix de la tâche Declined.

Pour le fichier de la figure 5.10, contrairement au modèle de notre approche, l'exécution de la tâche Accepted est effectuée directement sans donner le choix à l'utilisateur. Par conséquent, on n'aura pas une perte de temps après la tâche Order (temps de fin de order est 19h 33m et 54 s et le temps de début de Accepted est 19h 33m et 54s). Même comportement pour le résultat de la figure 5.11.

Par conséquent, l'approche par réseau de pétri ne permet pas à l'utilisateur de spécifier son choix mais elle l'effectue automatiquement selon les données du système.

Voici un tableau comparatif entre les résultats des deux approches (hybride et RDP) .

| Tâche | **Modèle hybride** | | | | | **Modèle RDP** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Temps d'exécution | temps début | temps de fin | temps d'attente | déclanchement | Temps d'exécution | temps début | temps de fin | temps d'attente | déclanchement |
| Order | 9s | 19h 31m 44s | 19h 31m 53s | 0s | Automatique | 9s | 19h 33m 45s | 19h 33m 54s | 0s | Automatique |
| Declined | \ | \ | \ | \ | \ | \ | \ | \ | \ | \ |
| Accepted | 4s | 19h 31m 57s | 19h 32m 1s | 4s | Utilisateur | 4s | 19h 33m 54s | 19h 33m 58s | 0s | Automatique |
| Receive book | 9s | 19h 32m 1s | 19h 32m 10s | 0s | Automatique | 9s | 19h 33m 58s | 19h 34m 6s | 0s | Automatique |
| Receive bill | 8s | 19h 32m 1s | 19h 32m 9s | 0s | Automatique | 8s | 19h 33m 58s | 9h 34m 7s | 0s | Automatique |
| Pay | 10s | 19h 32m 10s | 19h 32m 20s | 0s | Automatique | 10s | 19h 34m 7s | 19h 34m 17s | 0s | Automatique |

TAB. 5.4: Tableau comparatif des résultats de l'exécution des deux approches pour le choix de la tâche Accepted.

Dans ce tableau, la tâche Accepted dans le modèle hybride est déclenchée par l'utilisateur avec un temps d'attente égale à 4 secondes, c'est le temps passé par l'utilisateur pour effectuer son choix entre les deux tâches Accepted et Declined, cela illustre que notre modèle hybride est beaucoup plus flexible d'un côté et d'un autre elle permet un certaine contrôle entre les tâches en gardant la correction des dépendance de flux . Contrairement au modèle de RDP, il exécute le choix automatiquement et sans perte de temps mais ne donne aucune flexibilité et aucune possibilité à l'utilisateur de faire son choix et ainsi déclenche la tâche Accepted juste après la tâche order.

## 5.4 Conclusion

Dans ce chapitre nous avons présenté en détail notre approche hybride, et nous avons mis en œuvre un exemple d'un processus métier qui est le processus d'achat de livre présenté dans [33].

L'exécution correcte du modèle est un argument suffisant pour dire que l'idée de fermer et d'encapsuler les parties impératives dans les poches impératives et les parties déclaratives dans des poches déclaratives est faisable et donne un résultat d'exécution sans violation de contraintes et ainsi sans erreurs.

La flexibilité offerte par les poches déclaratives permet une spécification partielle au préalable et ainsi donne une certaine liberté à l'utilisateur de spécifier complètement le processus dans le temps d'exécution. Cette fonctionnalité qu'on ne peut pas trouver dans les modèle de réseaux de pétri qui ne donne aucune possibilité de changer le modèle dans le temps d'exécution.

Pour cette mise en œuvre nous avons utilisé l'environnement JBuilder X Entreprise pour Java, pour la représentation de graphes d'instances nous avons incorporé et utilisé l'API Java JGRAPH disponible sur le site http ://www.jgraph.com/.

L'implémentation de l'approche nous a permis de mieux comprendre la fonction d'automatisation de processus malgré les difficultés rencontrées. De toucher au domaine de théorie des graphes et d'implémenter quelques algorithmes, d'utiliser à notre manière les contraintes Template de DecSerFlow et de les représentés avec la théorie d'Allen.

# Conclusion générale

DANS ce travail, nous avons traité le problème du manque de contrôle dans les modèles déclaratifs de workflow. La flexibilité offerte par ces derniers permet à l'utilisateur de spécifier librement les processus. Néanmoins, le manque de contrôle dans ces spécifications peut causer plusieurs erreurs d'exécution ce qui entraîne des problèmes importants pour les entreprises et le bon déroulement de leur processus métiers.

Partant de ce contexte, l'objectif de notre travail est de proposer un modèle hybride de spécification de workflow qui puisse ajouter un certain degré de contrôle aux modèles déclaratifs de workflow et ainsi éviter les erreurs d'exécution qui peuvent survenir.

Le premier chapitre de ce mémoire est une étude bibliographique sur quelques terminologies de base utilisées dans la technologie workflow et la spécification de processus métier. Dans le deuxième chapitre nous avons abordé les deux méthodes de modélisation de workflow connus dans la littérature à savoir la méthode déclarative à base de langage LTL et la méthode impérative à base de réseau de pétri.

Nous avons abordé en détail dans le chapitre 3 un concept très intéressant pour l'incorporation de la flexibilité dans la spécification de workflow qui est le principe de poche de flexibilité. Cette dernière permet une spécification partielle au préalable et lors de l'exécution elle donnent la main à l'utilisateur pour spécifier les tâches de la poche en spécifiant complètement le processus.

Après une étude détaillée de deux méthodes de modélisation de workflow, nous avons opté dans le chapitre 4 qui est notre proposition à l'utilisation d'un modèle déclaratif comme point de départ de notre approche, et d'ajouter à ce modèle un certain degré de contrôle par l'ajout de points de contrôle et la création de poches impératives afin de tirer et cerner les parties impératives du modèle. Le principe

de poche nous à permis de penser à encapsuler les parties impératives (série ou parallèle) du modèle dans des poches impératives ce qui nous donne la possibilité de mieux contrôler le modèle. Dans le chapitre 5 nous avons montré la faisabilité de notre approche par une étude et une implémentation d'un cas d'un processus métier proposé par M. Pesic and W.M.P. van der Aalst [33].

Pour atteindre notre objectif, nous nous sommes basé sur les travaux de Shazia Sadiq et al [37] pour l'étude de concept de poche de flexibilité ce qui nous a donné l'idée de poches impératives. Van der Aalst and M. Pesic [40] pour l'étude d'un des langages déclaratif graphique DecSerFlow dont on a utilisé la syntaxe de ses formules de relations et de négations, Ruopeng Lu et al [27] l'application de l'algorithme de chemin consistant , et James F.Allen [9] pour la transformation des formules de relation du modèle en relations d'intervalle d'Allen afin de traiter le réseau de contraintes temporelles résultant.

- **Perspectives de recherche :**

Comme perspective de recherche, nous visons les points suivants :

1. Trouver un moyen pour une modélisation dynamique par RDP des poches impératives ce qui peut avoir comme résultats plusieurs instances de la poche et par conséquent du processus modélisé .
2. Avoir un algorithme d'exécution efficace qui puisse exécuter le modèle dans un temps minimum en suivant le flux dans les poches impératives selon la modélisation dynamique de RDP.

# Appendices

# Annexe A

## A.1 Exemple d'une modélisation d'un processus par le langage LTL

Pour illustrer l'utilisation du langage LTL dans la formalisation de processus (problèmes pratiques), nous utilisons un exemple de la cabine téléphonique (publiphone). Le travail consiste à modéliser les actions que l'utilisateur devra effectuer dans le processus de tentative de téléphoner jusqu'à la fin de la communication. Pour accomplir ce travail on va procéder comme suite :

1. ***Localiser tous les opérations effectuées par les deux acteurs (utilisateur, système)***.
   - L'utilisateur doit décrocher le téléphone.
   - Le système affiche insérer une carte.
   - L'utilisateur insère une carte.
   - Le système affiche le nombre d'unités restantes.
   - L'utilisateur compose son numéro de téléphone avec la possibilité de le corriger pour obtenir le bon numéro dans un délai de 2 secondes.
   - L'utilisateur communique avec son correspondant, tant qu'il lui reste des unités.
   - Si la carte est épuisée, l'utilisateur a 10 secondes pour la changer.
   - Lorsque la communication est terminée, l'utilisateur raccroche le téléphone.
   - Le système affiche retirer la carte.
   - L'utilisateur retire sa carte.
2. ***Formaliser les propositions atomiques***.
   - u-décrocher : l'utilisateur décroche le téléphone.
   - sys-affiche (message) :le système affiche un message.
   - u-insérer-carte : l'utilisateur insère une carte.
   - u-compser-no : l'utilisateur compose le numéro.
   - u-correction-no : l'utilisateur corrige le numéro.

- numéros-correct : le numéro est correct.
- u-communiquer : l'utilisateur communique avec son correspondant.
- u-changer-carte-vide : l'utilisateur change sa carte qui est épuisée
- u-raccrocher : l'utilisateur raccroche le téléphone.
- u-retirer-carte : l'utilisateur retire sa carte.
- ltl-téléphoner : établit la clause de l'activité téléphoner.

3. ***Formaliser la formule LTL.***

   u-decrocher $\wedge$ X sys-affiche(Insérer-carte) $\wedge$ XX[(u-insrer-carte $\wedge$ X sys-affiche (nb-units-restantes )) $\wedge$ (X u-composer-no $\vee$ F (u-correction-no U numros-correct)) $\wedge$ ((X u-communiquer $\vee$ F (u-changer-carte-vide U unites(carte) $>$ minimum)) U (u-raccrocher $\wedge$ u-retirer-carte))] $\Rightarrow$ ltl-telephoner.

## A.2 La sémantique de la formule

Voici une explication plus au moins détaille du processus de téléphoner :

L'utilisateur doit décrocher le téléphone, puis, dans l'état suivant (opérateur X), il doit insérer sa carte. Ensuite, il compose son numéro avec la possibilité (opérateur F) de le corriger. Si le numéro est correct, l'utilisateur pourra, alors communiquer avec la possibilité de changer sa carte. La communication dure jusqu'à ce qu'il raccroche le téléphone et retire sa carte. On a pu exprimer la notion de boucle à l'aide de l'opérateur U (jusqu'à). Par exemple, on corrige le numéro de téléphone un nombre infini de fois jusqu'à obtenir le bon numéro. En plus, l'opérateur X (suivant) précise la séquence d'évènements : l'utilisateur compose son numéro juste après que le système affiche le nombre d'unités restantes.

# Annexe B

## B.1 Exemple d'un réseau temporel inconsistant

Soit un réseau constitué d'un ensemble de variables (tâches) X=$\Big\{T1, T2, T3\Big\}$ et un ensemble de contraintes :
C=$\Big\{C_{12}, C_{13}, C_{23}\Big\}$ où
$C_{12} = T1 \Big\{b, m\Big\} T2$ et $C_{13}$= T1 $\Big\{s, eq\Big\}$ T3 et $C_{23}$ = T2 $\Big\{b, m\Big\}$ T3.
La définition de chaque contrainte ne cause pas de conflit puisque pour une instance de processus particulière exige une seule relation entre chaque paire de variables (T1 $\Big\{b\Big\}$T2).Cependant le réseau est inconsistant puisque pour $C_{12}$ et $C_{23}$ on peut inférer $C_{13}$'=T1$\Big\{b\Big\}$ $T3$ mais $C_{13}$' $\cap$ $C_{13} = \emptyset$ , ce qui fait qu'on ne peut pas trouver un scénario qui satisfait les trois contraintes $C_{12}$, $C_{13}$ et $C_{23}$ en même temps. Par conséquent le réseau est inconsistant. Par contre un réseau BPCN N est consistant si on peut trouver dans N un scénario consistant. [27]

# Annexe C

## C.1 Modéliser un service avec DecSerFlow

Dans ce qui suit on va présenter le cas de ***ACME traval company*** , une compagnie de voyage, pour illustrer l'utilisation de DecSerFlow [40]. La description de processus métier de service de voyage d'ACME est donnée comme suit :

- ACME voyage reçoit un itinéraire de karla (client).
- Après un contrôle d'itinéraire pour détecter les erreurs, le processus détermine quelle réservation à faire, envoyant des requêtes simultanées aux agences appropriés (ligne aérienne *airline*, ou hôtel *hotel* ) pour faire les réservation appropriés.
- Si une des réservations échoue, l'itinéraire sera fermé à l'aide de l'activité *compensate* et karla sera informé du problème
- ACME voyage attend la confirmation des deux réservations (airline et hotel).
- A la réception de la confirmation, ACME voyage informe karla de l'accomplissement réussite du processus et lui envoie le nombre de réservations confirmées et les détails de l'itinéraire final.
- Une fois que karla est informée de l'échec ou de succès de son itinéraire demandé, elle peut commander une autre demande de voyage.

La figure suivante représente un modèle DecSerFlow pour le processus métier d'ACME. Le schéma comprend onze activités (tâches).

$$\text{Tâches}=\left\{\begin{array}{c} receive, hotel, bookedHotel, \\ failedHotel, airline, bookedAirline, failedAirline, \\ creditCard, notifyBooked, compensation, notifyFailure \end{array}\right\}.$$

Sans avoir un contrôle fiable sur ces activités, il serait possible de les exécutées un nombre arbitraire de fois dans un ordre arbitraire, il serait aussi possible de ne pas exécuté aucune activité, pour empêcher un tel comportement, l'ajout des contraintes est primordiale pour le modèle de processus de service.

Une contrainte dans un service représente une règle que l'exécution de service doit vérifiée. Le modèle DecSerFlow utilise deux des trois types de contraintes déjà citées, les contraintes d'existences et les contraintes de relations.

Dans le modèle de la figure C.1, les contraintes d'existence (définit pour les activités unaires) sont représentées par les cardinalités au-dessus des activités, et les contraintes relations définit entre deux au plusieurs activités. Le tableau suivant illustre quelques contraintes utilisées dans le modèle de la figure C.1.

| Contraintes d'existence | |
|---|---|
| **Contrainte** | **Signification** |
| La cardinalité 1 au-dessus de l'activité **receive** | L'activité **receive** peut s'exécutée exactement une fois par une reqête du client |
| La cardinalité (0..1) au-dessus de l'activité compensation | L'activité compensation peut s'exécutée au plus une fois. |
| La cardinalité (0..1) au-dessus de l'activité notify failure | L'activité notify failure peut s'exécutée au plus une fois |
| La cardinalité (0..1) au-dessus de l'activité notufy booked | L'activité notufy booked peut s'exécutée au plus une fois. |
| La cardinalité (0..1) au-dessus de l'activité credit card | L'activité credit card peut s'exécutée au plus une fois. |
| Contraintes de relation | |
| **Contrainte** | **Signification** |
| response l'activité receive et hotel ou entre receive et airline | Après l'exécution de l'activité receive au moins l'une des activités (hotel ou airline) sera exécutée. |
| Precedence entre l'activité receive et hotel | L'activité hotel ne peut pas s'exécutée avant l'exécution de receive. |
| Precedence entre l'activité receive et airline | L'activité airline ne peut pas s'exécutée avant l'exécution de receive. |
| Response entre hotel et les deux activités failed hotel et booked hotel | Après l'exécution de l'activité hotel au moins l'une des activités failed hotel ou booked hotel s'exécute. |
| Response entre airline et les deux activités failed airline et booked airline | Après l'exécution de l'activité airline au moins l'une des activités failed airline ou booked airline s'exécute. |
| Precedence entre compensation et les deux activités failed hotel et failed airline | Après la terminaison de l'exécution des deux activités failed airline et failed hotel le service ACME peut fermer la réservation en exécutant l'activité compensation. |
| Not-response entre compensation et les deux activités hotel et ailine | Après l'exécution de compensation, il y aura pas d'exécution des activités hotel et airline car le service a fermé la réservation. |

Tab. C.1: Quelques contraintes utilisées dans le processus métier d'ACME .

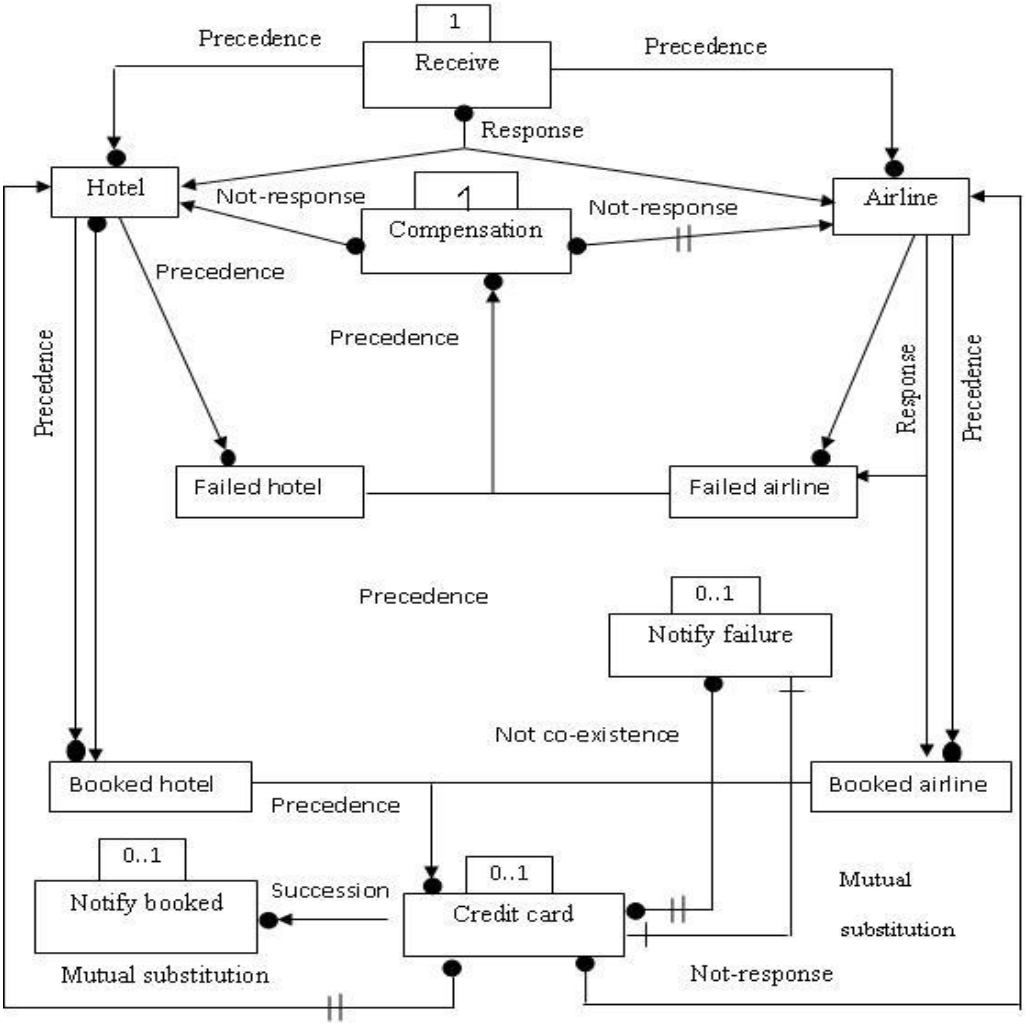



FIG. C.1: DecSerFlow - Acme Travel Company [40].

# Annexe D

## D.1 Représentation des formules de relations avec les relations d'intervalle d'Allen

Voici un tableau décrivant pour chaque formule de relation du langage graphique DecSerFlow sa relation équivalente en intervalle d'Allen.

| **Formule de relation** | **Relation d'intervalle d'Allen** |
|---|---|
| **Responded_Existence** Existence(A)→existence(B),♦(A)⇒♦(B) | A$\{eq, o, s, f\}$B |
| **Co_existence** Existence(A)↔existence(B) | A$\{eq, s, f, o\}$ B |
| **Response**❑(activity=A→existence(B)) | A$\{b, m\}$B |
| **Precedence**(Existence(B)→( !=(activity==B)U activity==A)) | B $\{a, mi\}$ A |
| **Succession** : (A_response_B(A,B) ∧ A_precedence_B(B,A)) | A$\{b, m\}$B ou B$\{a, mi\}$A |
| **Alternate_response** (A_response_B(A,B) ∧$B\_always\_between\_A(A, B)^*$) | A$\{b, m\}$B ou B$\{a, mi\}$A |
| **Alternat_precedence** A_precedence_B(A,B) ∧ $B\_always\_between\_A(B, A)^*$) | A$\{b, m\}$B ou B$\{a, mi\}$A |
| **Alternate_succession** (A_alternate_precedence_B(A,B) ∧ A_alternate_response_B(A,B)) | A$\{b, m\}$B ou B$\{a, mi\}$A |
| **Chain_Response** (A_response_B(A,B)∧ ❑ (activity==A → ❍ (activity==B))) | A$\{m\}$B |
| **Chaine_precedence** ❑ ((❍(Activity==B) → activity==A) | B $\{mi\}$A |
| **Chaine_succession** (Chain_A_response_B(A,B)∧ Chain_A_precedence_B(A,B)) | B $\{mi\}$A et A $\{m\}$B |

Tab. D.1: Représentation des formules de relations avec les relations d'intervalle d'Allen.

# Bibliographie


[1] W.V.D Alast. A class of petri nets for modeling and analyzing business processes. *Computing Science Reports 95/26, Eindhoven University of Technology*, 1995.

[2] W.V.D Alast. Verification of workflow nets. *Lecture notes in computer science*, June 1997.

[3] W.V.D Alast. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers.*, 1998.

[4] W.V.D Alast. Workflow verification : Finding control-flow errors using petri-net-based techniques. *Eindhoven University of Technology ,The Netherlands*, 2000.

[5] W.V.D Alast and H.M.W. Verbeek. A. Hirnschall. An alternative way to analyze workflow graphs. *Advanced Information Systems Engineering, 14th International Conference*, (535 - 552), May 2002.

[6] W.V.D Alast and O.perrin H.Skaf F.Charoy, D.Grigori. Workflow concepts et techniques.

[7] W.V.D Alast and G.J. Houben. K.M. van Hee. Modelling and analysing workflow using a petri-net based approach. *Eindhoven University of Technology, The Netherlands.*, 1994.

[8] W.V.D Alast and A.H.M. ter Hofstede. Workflow patterns : On the expressive power of (petri-net-based) workflow languages. *Eindhoven University of Technology - The Netherlands, Queensland University of Technology -Australia*, 2002.

[9] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM, The University of Rochester*, (832-843), novembre 1983.

[10] D. Berthier and M. Maurice-Demourioux C. Morley. Enrichissement de la modélisation des processus métiers par le paradigme des systèmes multi agents. *INT/GET Groupe des Écoles des Télécommunications , France*, 2005.

[11] A. Bobbio. System modelling with petri nets. *Istituto Elettrotecnico Nazionale Galileo Ferraris Strada delle Cacce 91,10135 Torino, Italy*, (102-143), 1990.

[12] A. Dicky. Graphes et algorithmes résumé de cours. *Conservatoire National des Arts et Métiers*, 1998.

[13] R. Eshuis and J. Dehnert. Reactive petri nets for workflow modeling. *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets (ICATPN 2003), Eindhoven, The Netherlands*, (296 -315), June 2003.

[14] E.Sivaraman and M. Kamath. On the use of petri nets for business process modeling. *School of Industrial Engineering and Management, Oklahoma State University.*, 2002.

[15] S. Haddad et J-M. Bernard. Les reseaux reguliers : Specification et validation par le logiciel ARP. *Exhibition on Software Engineering*, (83-93), 1986.

[16] T. Vantroys et P. Yvan. Un système de workflows flexible pour la formation ouverte et à distance. *Université des Sciences et Technologies de Lille*, Janvier 2002.

[17] M.S. Evangelista. *Méthodes et Outils de Vérification pour les Réseaux de Petri de Haut Niveau.Application à la Vérification de Programmes Ada Concurrents.* PhD thesis, Conservatoire National des Arts et Métiers, Décembre 2006.

[18] D. Fahland. Synthesizing petri nets from LTL specifications, an engineering approach. *Humboldt-UniversitÄat zu Berlin, Germany*, September 2007.

[19] D. Fahland. Towards analyzing declarative workfows. *Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Unter den Linden 6, 10099 Berlin, Germany*, 2007.

[20] D. Florent. *Application de Workflow, mise en place dance une entreprise industrielle.* PhD thesis, Ecole de Mines de Paris. France, Novembre 2004.

[21] E.A. González and N. Debnath W.W Smari. Toward flexibility of workflow management systems based on task requirement classification. *The 2002 International Conference on Information Systems and Engineering*, (210-219.), 2002.

[22] K. Grigorova. Process modelling using petri nets. *International Conference on Computer Systems and Technologies, CompSysTech, University of Rousse*, 2003.

[23] M. Hamme and J. Champy. What is reengineering ? information week. 372(5)(5) :10,14, 18, 20, 24., may 1992.

[24] J.Wang. Petri nets for dynamic event-driven system modeling. *Handbook of Dynamic System Modeling.*, 2007.

[25] Z. Sbaï K. Barkaoui, R. Ben Ayed. Workflow soundness verification based on structure theory of petri nets. *International Journal of Computing and Information Sciences, Vol. 5, No. 1*, Avril 2007.

[26] K.M.V. Hee L. Zerguini. A new reduction method for the analysis of large workflow models. *FTC27 workshop, Toyama, Japan*, 1992.

[27] R. Lu and G. Governatori S. Sadiq, V. Padmanabhan. Using a temporal constraint network for business process execution. *School of Information Technology and Electrical Engineering The University of Queensland*, 2006.

[28] D.C. Ma and M.E. Orlowska J.Y.C. Lin. On grouping of activities in workflow management systems. *The University of Queensland Brisbane, Australia.*, 2007.

[29] P. Mangan and S. Sadiq. On building workflow models for flexible processes. *School of Computer Science and Electrical Engineering , The University of Queensland, Australia*, 2002.

[30] P.J. Mangan and S. Sadiq. A constraint specification approach to building flexible workflows. *Journal of Research And Practice In Information Technology*, (21-39), 2003.

[31] W.V.D Alast M.Pesic, H. Schonenberg. Declare :full support for loosely-structured processes. *Eindhoven University of Technology The Netherlands*, 2007.

[32] E. Oren and A. Haller. Formal frameworks for workflow modeling. *DERI Technical Report, National University of Ireland, Galway.*, Avril 2005.

[33] M. Pesic and W.V.D Alast. A declarative approach for flexible business processes management. *Lecture notes in computer science, P.O.Box 513, NL-5600 MB, Eindhoven, The Netherlands.*, 2006.

[34] D. Qiang. *Workflow Simulation for international trade.* PhD thesis, University of Auckland, 2002.

[35] W. Mike S. Khodakaram. Petri net-based modelling of workflow system : An overview. *European Journal of operationel Research 134, (664-676),Lancaster University*, August 2000.

[36] S. Sadiq and M. Orlowska W. Sadiq. Pockets of flexibility in workflow specification. *School of Computer Science and Electrical Engineering, The University of Queensland*, 2001.

[37] S.W. Sadiq and W. Sadiq M.E. Orlowska. Specification and validation of process constraints for flexible workflows. *School of Information Technology and Electrical Engineering, The University of Queensland, Qld 4072, Australia Corporate Research, SAP Australia Pty Ltd, Brisbane, Australia*, May 2004.

[38] E.M. Clarke T. Yoneda, H. Schlingloff. Efficient timing verification based on one-safe time petri nets and a real-time logic. *FTC27 workshop, Toyama, Japan.*, July 1992.

[39] M. TISSOT. Rapport de stage, environnements d édition de workflow. *projet Opéra - INRIA.*, Août 2000.

[40] W.M.P. van der Aalst and M. Pesic. DecSerFlow : Towards a truly declarative service flow language. *Lecture Notes in Computer Science*, (1-23), 2006.

[41] I. Vondrak. Business process modeling and workflow automation. *VSB-Technical University of Ostrava*, 2001.

[42] K. van Hee W.V.D Alast. Workflow management models, methods and systems. *Université de technologie Eindhoven , Hollande*, dec 2000.